Project Portfolio

Daniel Bergeron

COMP IV Sec 203

Spring 2023

PS0 Hello World with SFML

PS1 Linear Feedback Shift Register and Image Encoding

- PS1a Linear Feedback
- PS1b Image Encoding

PS2 Sokoban

PS3 Pythagoras Tree

PS4 Checkers

PS5 DNA Sequence Alignment

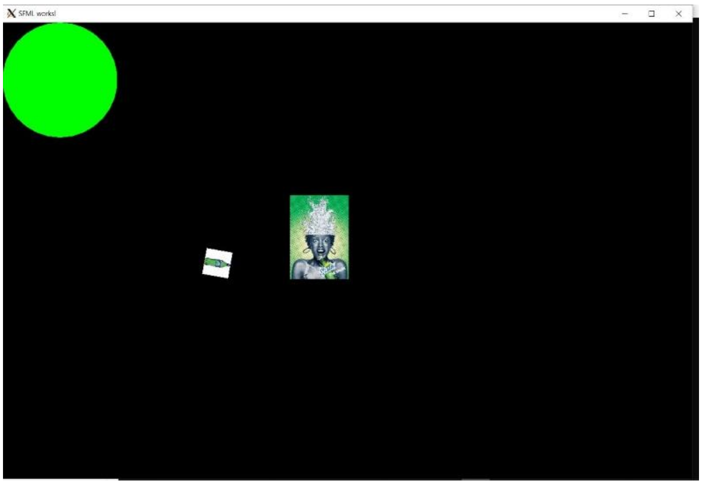
PS6 Random Writer

PS7 Kronos Time Clock

Time to complete: 20hrs

Table of Contents

PS0 – Hello World



Window Output 1

Program Discussion

PS0- Hello World, was an introductory program to the Computing IV class, as well as an introduction/tutorial for the SFML library. SFML is an API that assists users in many aspects, for this assignment I used SFML to deploy PNG images into a visual environment. The environment I used to launch windows from is called XLAUNCH. I took the image out of its location and loaded it into an object called a sprite which is light weight and easy to interact with object. I was then able to play around with the object's methods, to resize and move the object around the screen. It should be noted that at the beginning of this project I thought a sprite was an actual sprite drink, hence the theme. Using the sf::Keyboard class I was able to give the user the ability to change the image attributes. When they hit a certain key, the sprite would move, and its size would increase. This program was built out of the tutorial code in the SFML library, hence the green circle which is supposed to represent the world playing into the title of Hello World.

What I Learned

This program acted as a sort of warm up for the heavy amount of coding I would be doing this semester. The main way ps0 did this was re-introducing me to C++ and it forced me to interact with an API that I had very little experience with. To be pedantic the exact items I learned how to interact with were the sf::RenderWindow class, the sf::Keyboard class, the sf::Sprite, the sf::CircleShape and the sf::sprite class. While none of what I accomplished and learned in this program taught me anything new regarding the C++ program. The program introduced me to import aspects of SFML that I would use throughout the semester.

Issues

In this assignment there was not much in the way of classical programing bugs. However, I had issues getting my window to work properly, initially I was experiencing segmentation faults every time I opened a new window. Eventually through some debugging I was able to get my window to work properly. I was no longer experiencing any issues with segmentation faults, but I was never able to figure out why setting up vertical sync in terminal kept failing. This had no effect on my program, so I ignored the issue and continued with the assignment.

Code

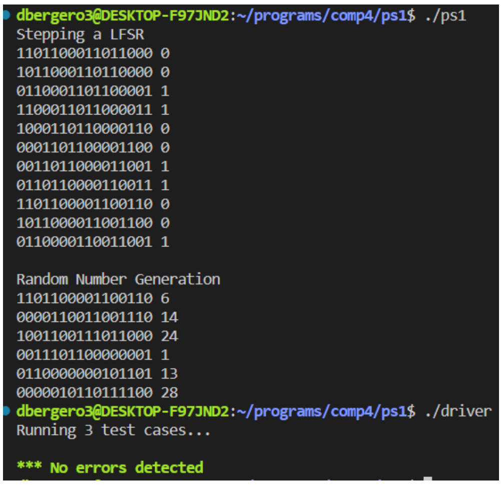
Main

```
/*
*Daniel Bergeron
*COMP IV
*helloworld-ps0
*Dr. Yelena Rykalova
*1/24/2023
*Program tests the smlf library
*
*/
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

int main()
{
        //keystroke detection variables
        float x = 400, y = 400, currentXScale = .1, currentYScale = .1 ;

        sf::RenderWindow window(sf::VideoMode(1200, 1200), "SFML works!");
        window.setFramerateLimit(60);
        sf::CircleShape shape(100.f);
        shape.setFillColor(sf::Color::Green);

        //load image of sprite
        sf::Texture texture;
        sf::Texture texture2;
        if(!texture.loadFromFile("sprite.png")){
                return EXIT_FAILURE;
        }
        if(!texture2.loadFromFile("sprite2.png")){
                return EXIT_FAILURE;
        }

        sf::Sprite sprite;
        sf::Sprite sprite2;

        sprite.setTexture(texture);
        sprite.setPosition(x, y);
        sprite.setScale(currentXScale, currentYScale);

        sprite2.setTexture(texture2);
        sprite2.setPosition(x + 100, y - 100);
        sprite2.setScale(currentXScale, currentYScale);

        while(window.isOpen())
        {

        sf::Event event;
        while(window.pollEvent(event)){
                if(event.type == sf::Event::Closed)
                        window.close();
        }

        window.clear();
        if(sf::Keyboard::isKeyPressed(sf::Keyboard::Left))
        {
                sprite.move(-10.00,0);
                currentXScale += .01;
                sprite.setScale(currentXScale, currentYScale);
        }
        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Right)){
                sprite.move(10.00,0);
                currentXScale += .01;
                sprite.setScale(currentXScale, currentYScale);
        }
        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Up)){
                sprite.move(0, -10.00);
        currentYScale += .01;
        sprite.setScale(currentXScale, currentYScale);
        }
        else if(sf::Keyboard::isKeyPressed(sf::Keyboard::Down)){
                sprite.move(0, 10.00);
                currentYScale += .01;
                sprite.setScale(currentXScale, currentYScale);
        }
        sprite.setRotation(sprite.getRotation() + 20.00 );

        window.draw(shape);
        window.draw(sprite);
        window.draw(sprite2);
        window.display();

        }
return 0;
}

```

PS1A – Linear Feedback Shift Register

Program Discussion

The Goal of this assignment was to create a Linear Feedback shift register (LFSR), the idea is to construct one of these registers and then use it to as a pseudo random number generator. To emulate an LFSR I implemented a class that uses a binary string to construct an object that would represent an LSFR with three tap points. The tap points are the area in the register which is accessed and then XORed together to create the pseudo random aspect of an LFSR. This object would then be used in my main routine to generate these random values. I was fully able to create the LFSR. I was successful in my approach to this project because I was able to channel OOP principles to generate an effective class system to handle the register. Another Important aspect this program introduced, was testing, and it is at this point that I must introduce the boost library, this API allows a programmer to efficiently test their code for any issues. I had two drivers for this program, one for my main routine, and one for my boost test cases. Boost allowed me to really test my code and by the end of the project I was able to pass all my cases, and my main routine worked as it was supposed to.



```
dbergero3@DESKTOP-F97JND2:~/programs/comp4/ps1$ ./ps1
Stepping a LFSR
1101100011011000 0
1011000110110000 0
0110001101100001 1
1100011011000011 1
1000110110000110 0
0001101100001100 0
0011011000011001 1
0110110000110011 1
1101100001100110 0
1011000011001100 0
0110000110011001 1

Random Number Generation
1101100001100110 6
0000110011001110 14
1001100111011000 24
0011101100000001 1
0110000000101101 13
0000010110111100 28
dbergero3@DESKTOP-F97JND2:~/programs/comp4/ps1$ ./driver
Running 3 test cases...

*** No errors detected
```

Main Routine & Test Case Output 1

Key Elements Discussion

One of the main features of this program that led to my success, was its object-oriented approach. The area this was applied was in creating my virtual, LSFR. I took the idea of an LSFR and wrapped It into a class. This class was responsible for building my virtual LSFR. Essentially my class gave the user an interphase to interact with that acted as a LSFR. From the outside looking in it would be very difficult to tell that you weren't dealing with an actual LSFR but rather something designed to emulate it. I accomplished this with a key principle of OOP called encapsulation essentially, I took a set of methods and variables and wrapped them into one concise thing that the user does not get to see. If you are looking at my main routine all you see is the object and the methods and variable that I allow you to see. This level of abstraction made the main routine in my code simple. In addition, this program also contained a header, that held all the important information about my LSFR class but abstracted away any information about implementation. This allowed me to create a very easy to read program, with a wide range of functionality. For example, there was a step operation that took the inner representation of the LSFR and moved it left once and then generated a new bit. There was also a generate function that generated a random number for the user to use. Since the eternal representation of this code is in an area where the user can' t reach it makes it very hard to break code. Behind all this the data which the user was manipulating was my own personal take on the best internal representation. For this I used another object called a bit set which acted as an interphase between the me and what in my head was a 16-bit place in memory. Me being the user in this situation I could only interact with the object so anything under the hood did not concern me. Also, in this assignment I overloaded the extraction operator to output my data in way that would make since to the idea of a LSFR register. This plays into a different component of OOP called polymorphism, or the idea that a function can mean different things depending on how it is applied. This paradigm of programming that I implemented for this project allowed me to complete this project.

What I Learned

In this assignment, I learned and improved a great deal. Firstly, I learned how to use the boost testing library for my code. I also learned what an LFSR was, to create an object to understand it I had to have a great understanding of how the machine worked. This was also my first experience using the bit set library which was a great asset for me when I was trying to come up with internal representation of the LFSR. As a programmer this program improved my understanding of OOP design and how powerful it is at simplifying code.

Issues

In this program, I faced potentially my most tragic mistake as a programmer. I didn't back up my files and an error caused me to accidentally delete my entire program. I had to rush a couple hours before the due date to finish the assignment. This became one of my fastest ever programmed assignments. From this I learned to always back up code. One good way is creating a GitHub repository.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -c -g -Og -Wall -Werror -pedantic
DEPS = FibLFSR.hpp
OBJ = FibLFSR.o main.o
OBJ_A = FibLFSR.o test.o
LIBS = -lboost_unit_test_framework

all: ps1 driver

ps1: $(OBJ) $(DEPS)
	$(CC) $(CFLAGs) -o ps1 $(OBJ)
driver: $(OBJ_A) $(DEPS)
	$(CC) $(CFLAGs) -o driver $(OBJ_A) $(LIBS)
test.o: test.cpp
```

```
	$(CC) $(CFLAGS) -c test.cpp -o test.o
main.o: main.cpp
	$(CC) $(CFLAGS) -c main.cpp -o main.o
FibLFSR.o: FibLFSR.cpp FibLFSR.hpp
	$(CC) $(CFLAGS) -c FibLFSR.cpp -o FibLFSR.o
clean:
	rm -f $(OBJ) main.o test.o ps1 driver
```

Main.cpp

```
#include "FibLFSR.hpp"

int main(int argc, char* argv[]){
FibLFSR test1("0110110001101100");
int newNum = 0, newNum2 = 0;
cout << "Stepping a LFSR\n";
for(int i = 10; i >= 0; i--){
        newNum = test1.step();
        cout << test1 << " " << newNum << endl;
}

cout << endl;
cout << "Random Number Generation\n";
FibLFSR test2("1100011011000011");
for (int i = 0; i < 6; i++) {
        newNum2 = test2.generate(5);
        cout << test2 << " " << newNum2 << endl;
        }
return 0;
}
```

FibLFSR.hpp

```
#include <bitset>
#include <string>
#include <iostream>
using namespace std;

class FibLFSR {
 public:
FibLFSR(std::string seed); // constructor to create LFSR with
// the given initial seed
int step(); // simulate one step and return the
// new bit as 0 or 1
int generate(int k); // simulate k steps and return
// k-bit integer
friend ostream& operator<<(ostream& out, FibLFSR reg);
```

```

 private:
        std::bitset<16> data;
};
```

FibLFSR.cpp

```cpp
#include "FibLFSR.hpp"

FibLFSR::FibLFSR(std::string seed){
        if(seed.size() != 16){
                cout << "ERROR: seed must have size of 16\n" ;
                return;
        }
std::bitset<16> newData(seed);
data = newData;
}

ostream& operator<<(ostream& out, FibLFSR reg)
{
        out << reg.data;
        return out;
}

int FibLFSR::step(){
        int tap1 = 15, tap2 = 13, tap3 = 12 , tap4 = 10;
        bool newVal = false;
        std::bitset<16> newData = data;
for(int i = 15; i > 0; i--){
        newData[i] = data[i -1];
}
        newVal = data[tap1] ^ data[tap2];
        newVal = newVal ^ data[tap3];
        newVal = newVal ^ data[tap4];
        newData[0] = newVal;

data = newData;
return newVal;
}

int FibLFSR::generate(int k){
        int value = 0;

        for(int i = 0; i < k; i++ ){
                value *= 2;
                value += step();
        }
        return value;
}
```

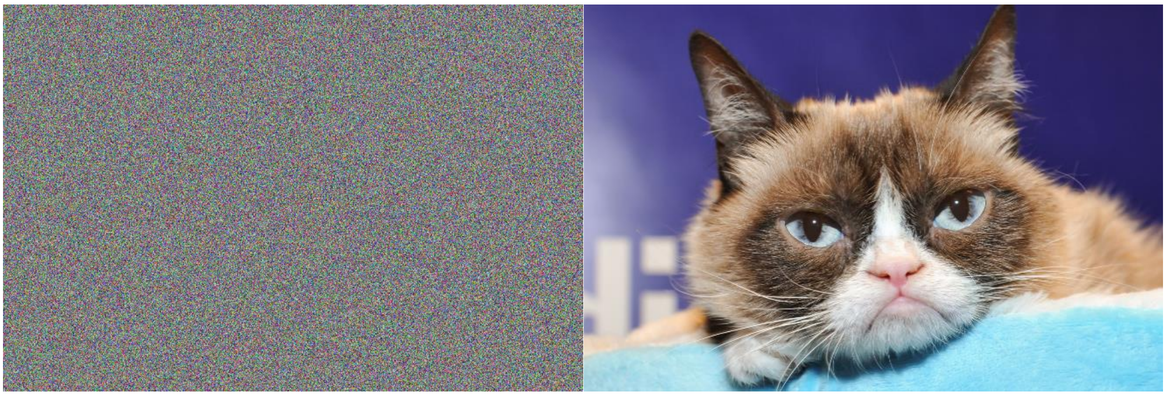
test.cpp

```cpp
// Dr. Rykalova
// test.cpp for PS1a
// updated 1/31/2020

#include <iostream>
#include <string>

#include "FibLFSR.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE TEST
#include <boost/test/unit_test.hpp>


BOOST_AUTO_TEST_CASE(sixteenBitsThreeTaps) {

FibLFSR l("1011011000110110");
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 1);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 0);
BOOST_REQUIRE(l.step() == 1);

FibLFSR l2("1011011000110110");
        BOOST_REQUIRE(l2.generate(9) == 51);
}

BOOST_AUTO_TEST_CASE(ZeroTest) {
        FibLFSR l("0000000000000000");
        for(int i = 0; i < 1000; i++){
                BOOST_REQUIRE(l.step() == 0);
        }
}

BOOST_AUTO_TEST_CASE(OnesTest) {
        FibLFSR l("1111111111111111");
        BOOST_REQUIRE(l.generate(11) == 0);
        BOOST_REQUIRE(l.generate(12) != 0)}
```

PS1B – Image Encoding

Program discussion

As shown by the header this program is built on the original program and introduced an interesting use of the LFSR. That use is the encoding and decoding of an image's pixels using the pseudo random generation of the LFSR. This program forced me to interact with the SFML library, which as discovered in PS0 is a great way to manipulate and use images. With this program I was able to take a PNG image and scramble its pixels. Then I could take that scrambled image, send it back through the program and get the same image back. Thus, demonstrating the powers of the LFSR for encoding and decoding. I was able to accomplish this with a cat image.



Window Output 2

Key Elements Discussion

The key elements of this program are the same as the previous assignment since it is built of it. This Program still implements the OOP design elements of part A. I did have to interact more with SFML more, which is something that employs Object-Oriented principles. I had to interact with the image object far more in this program. This is because I had to use the LFSR on each individual pixel in the PNG. I changed the color of each of the images based on the number in the LFSR, and that is what scrambles the image.

What I Learned

In this program I learned to better understand the object I created I the previous assignments and how to better use it. The heavy focus of this program was on the main routine since it interacted with SFML and is responsible for the interactions between my object and the ones in the SFML library. From this assignment I learned more about the draw loops in main that maintain the windows that SFML uses. But most of all this program taught me that images are data and data can be manipulated in ways that help or hurt the user depending on what your objective is.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
DEPS = FibLFSR.hpp
OBJ = FibLFSR.o PhotoMagic.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system

all: ps1B

ps1B: $(OBJ)
	$(CC) $(CFLAGS) -o ps1B $(OBJ) $(LIBS)
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) ps1B
```

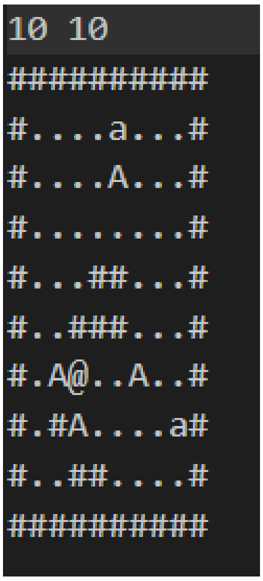
PhotoMagic.cpp

```
// pixels.cpp:
// using SFML to load a file, manipulate its pixels, write it to disk
// Fred Martin, fredm@cs.uml.edu, Sun Mar  2 15:57:08 2014
// g++ -o pixels pixels.cpp -lsfml-graphics -lsfml-window

#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include "FibLFSR.hpp"
#include <vector>

void transform( sf::Image& image, FibLFSR* password);

int main(int argc, char* argv[])
{
	string inputFileName, outputFileName, seed;
	cin >> inputFileName >> outputFileName >> seed;
	sf::Image input;
	if (!input.loadFromFile(inputFileName))
		return -1;
	sf::Image output;
	if (!output.loadFromFile(inputFileName))
		return -2;
	if(seed.size() != 16)
		return -3;
	FibLFSR password(seed);
	output = input;
```

```
        cout << "input in!\n";

        transform(output, &password);
        cout << "Transformation Complete\n";

        sf::Vector2u size1 = input.getSize();
        sf::RenderWindow window1(sf::VideoMode((size1.x), (size1.y)), "Input");
        sf::Vector2u size2 = output.getSize();
        sf::RenderWindow window2(sf::VideoMode((size2.x), (size2.y)), "output");

        sf::Texture texture1;
        texture1.loadFromImage(input);
        sf::Texture texture2;
        texture2.loadFromImage(output);

        sf::Sprite sprite1;
        sprite1.setTexture(texture1);
        sf::Sprite sprite2;
        sprite2.setTexture(texture2);

        while (window1.isOpen() && window2.isOpen()) {
                sf::Event event;
        while (window1.pollEvent(event)) {
                if (event.type == sf::Event::Closed){
                        window1.close();
                }
        }
        while (window2.pollEvent(event)) {
                if (event.type == sf::Event::Closed){
                        window2.close();
                }
        }
        window1.clear(sf::Color::White);
        window1.draw(sprite1);
        window1.display();
        window2.clear(sf::Color::White);
        window2.draw(sprite2);
         window2.display();
        }

        if (!output.saveToFile("output-file.png"))
                return -1;
        return 0;
}

void transform( sf::Image& image, FibLFSR* password){
        sf::Color newColor;
        int8_t redVal, greenVal, blueVal;
        sf::Vector2u size = image.getSize();
        for(unsigned int x = 0; x < size.x; x++){
                for(unsigned int y = 0; y < size.y; y++){
                        redVal = password->generate(8);
                        greenVa      l = password->generate(8);
                        blueVal = password->generate(8);
                        newColor = image.getPixel(x , y);
                        newColor.r = newColor.r ^ redVal;
                        newColor.g = newColor.g ^ greenVal;
                        newColor.b = newColor.b ^ blueVal;
                        image.setPixel(x, y, newColor);
                }
        }
return;
}
```

PS2A – Sokoban

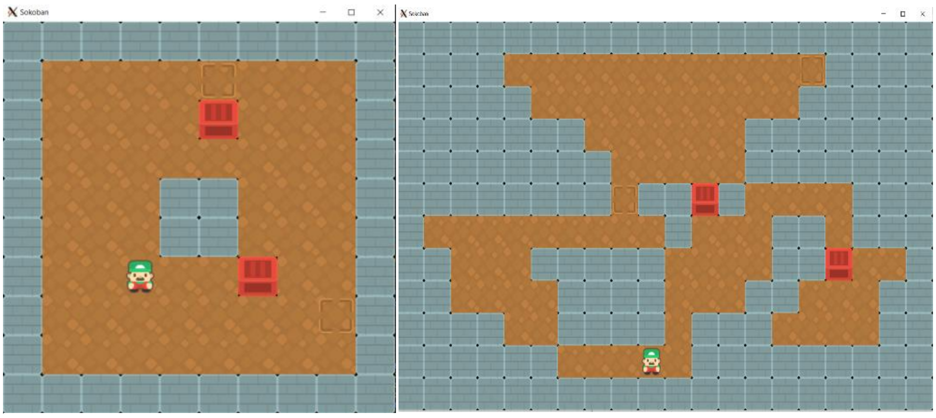
Program Discussion



```
10 10
##########
#....a...#
#....A...#
#........#
#...##...#
#..###...#
#.A@..A..#
#.#A....a#
#..##....#
##########
```

Input 1

This program is one of a two-part program. The goal of this program is to load and display a Sokoban map using SFML. This project was challenging because it involved a lot of moving parts. I was able to load and display the map, but due to lack of foresight I was unable to see that some decisions I made in this program would have a very negative effect on my next version of this project. PS2A focuses on taking in input, storing it, and the using that internal representation to generate a game map in an interactive window. I was able to accomplish this in a series of steps. The first being creating a place to store data, but to do that I needed to know where that data was coming from. Lucky enough that part was given. The data came from the file stream in the form of character symbols which were place holders for the types of objects that should be used in their place. This was useful because the layout of the input data gave me positional data on where objects should be on the game map, as well as the size of the game map. To gain the benefits of the input's layout, I needed to load it directly into the object I created. So, in my class definition I overloaded the insertion operator to take a file stream and put its content directly into my object. In my main function I got the file and instantiated my Sokoban object. It is here that I made a critical error that would make this project much more difficult than it needed to be. The mistake was not building in a static map background into the object. In either case, the constructor, or the insertion I could have created a static background for the object. This doesn't seem like a big deal for this part because the entire display is going to be static. And, while this is true, it was not good foresight to leave mix the static elements with the future would be dynamic items. This also would have been an easy fix. In my class definition, I used a matrix of special objects called block, to store each symbol. I chose a matrix so I could mirror the grid layout of the input file and by extension the game board. I could have made a separate constant matrix called background that loads the entire background floor tiles, and the walls if I wanted to be more pedantic. Moving past this design oversight the matrix in combination with my block class created a nice interphase for my main routine to interact with. It's also important note, that the Sokoban class inherits from the sf::drawable class. This feature allowed me to override SFMLs draw function and teach it how to draw my object. This was difficult to understand at first, but it took a lot of stress off me. Due to the fact that this function and this function alone were responsible for all the drawing happening in the main routine, I essentially didn't have to worry about the SFML aspect of the program. While making my methods I left everything visual up to the draw function. In conclusion my matrix system did allow my program to handle a diverse set of maps. After this teaching my program how to handle and display input, I needed only teach it how to play the game. Decisions I made in this assignment's implementation translated for better or worse into the following program.



Window Output 3

Key Elements Discussion

To create this program, I had to use some powerful programming tools. Firstly, to construct my matrix data structure I implemented the C++ vector library. Then with my vector objects I created a vector of vectors that mimicked how an actual matrix would work. I used the idea of row major order to implement this matrix. I also used a vector that holds class objects called gameobjects. I think of these game objects as actual blocks that represent an in-game tile. Using the insertion operator, I gathered symbols from the input stream, forged them into blocks and inserted them into this vector. From this vector I moved the objects into the matrix, and destroyed the vector. These blocks keep track of their own position, so when finally put them into my matrix they are in the correct position. Those are the two main data structures I used In this program. The next important attribute of this program is its OOP elements. The program uses inheritance to take information from the parent class sf::drawable and deliver it to Sokoban class. This is then used in a polymorphic way regarding the draw function. This Sokoban class is further abstracted by the GameObject class who handles the more precise interactions between blocks. All of this is encapsulated away from the user inside my classes. Somethings like the drawable class are even abstracted away from me. Another interesting aspect of this program is it is the first of my programs to employ a linter. Linter is a program that goes through code and checks for stylization errors. The implementation of a liner helped me produce neat and organized code.

What I Learned

This program introduced two new things to me. The first of these things is the draw function. This was the first time I used inheritance and polymorphism in this way. The SFML API abstracted away a lot of details, so I had to try to understand things from a top-down perspective, where things were abstracted away. It took me a while in this project to figure out how the draw function interacted with the API. The second thing that was completely new to me was the Linter. Before this I didn't even know these scripts existed, let alone how they benefited my time. Instead of searching through hundreds of lines of code for formatting errors I had the Linter tell me exactly where and what I need to fix. In addition, it gave me tips to do this. This program also built off a lot of things I am already familiar with, the most significant being OOP. Out of all the programs I’ve done this year this one increased my understanding the most.

Issues

In this code I couldn’t figure out how to implement the linter in my makefile so I used it as a VScode extension.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = Sokoban.o main.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system
DEPS = Sokoban.hpp

all: test

test: $(OBJ)
	$(CC) $(CFLAGS) -o test $(OBJ) $(LIBS)
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) test
```

Main.cpp

```
/* "Copyright [2023] <Daniel Bergeron>" */
#include "Sokoban.hpp"

// Function abstracts the grunt work of
// making the visuals away from main
// pre: Is passed a valid Sokoban object
void generateGameMap(Sokoban& level);

// pre: Passed valid texture and int pair by refrence
// post: pair will be given new value
void getTexturePixelSize(sf::Texture texture, pair<int, int>& cor);

int main() {
  // Simple driver to load and test maps
  Sokoban lv;
  ifstream fp;

  // test1
  fp.open("level1.txt");
  fp >> lv;
  generateGameMap(lv);
  fp.close();

  // test2
```

```
    fp.open("level2.txt");
    fp >> lv;
    generateGameMap(lv);
    fp.close();

    // stub test
    lv.movePlayer(noMove);

    return 0;
}

void generateGameMap(Sokoban& level) {
    sf::Texture texture, ground;;
    sf::Sprite sprite, floor;
    sf::Event event;
    sf::Vector2f location;
    pair<int, int> textureSize;
    GameObject block;
    bool drawn = false;

    // gets pixel sizes & creats a window that will fit all gamObjects
    texture.loadFromFile("sokoban/images/environment_03.png");
    getTexturePixelSize(texture, textureSize);
    sf::RenderWindow window(sf::VideoMode(textureSize.first * level.getWidth(),
    textureSize.second * level.getHeight()), "Input");

    // Game Loop, draw and maintain map
    while (window.isOpen()) {
        while (window.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window.close();
            }
        }
        if (drawn == false) {
            // pulls blocks from matrix to be drawn
            for (int y = 0; y < level.getHeight(); y++) {
                for (int x = 0; x < level.getWidth(); x++) {
                    block = level.getGameObject(y , x);
                    setDrawPos(location, block, textureSize);
                    // Special Case Player Block, ground needs to be drawn first
                    if (block.getTypeObject() == player) {
                        ground.loadFromFile("sokoban/images/ground_01.png");
                        floor.setTexture(ground);
                        floor.setPosition(location);
                        window.draw(floor);
                    }

                    // Draws and updates window
```

```
                          level.setSprite(block);
                          sprite = level.getSprite();
                          sprite.setPosition(location);
                          window.draw(sprite);
                          window.display();
                    }
             }
             drawn = true;
       }
 }
}

void getTexturePixelSize(sf::Texture texture, pair<int, int>& cor) {
 sf::Vector2u pixels;
 texture.loadFromFile("sokoban/images/environment_03.png");
 pixels = texture.getSize();
 cor.first = static_cast<int>(pixels.x);
 cor.second = static_cast<int>(pixels.y);
 return;
 }

```

Sokoban.hpp

```
/* "Copyright [2023] <Daniel Bergeron>" */
#include <fstream>
using std::ifstream;
#include <utility>
using std::pair;
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <vector>
using std::vector;
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

// enum describes set of object types and move types
enum typeObject{nill, player, wall, floor, crate, storage_space, stored};
enum moveSet{noMove, up, down, left, right};

class GameObject {
 private:
    int x;
    int y;
    typeObject object;
 public:
    // constructors
    GameObject() : x(0), y(0), object(nill) {}
    GameObject(int _x, int _y, typeObject _object);
    // accessors
    int getXlocation(void) const {return x; }
    int getYlocation(void) const {return y; }
    typeObject getTypeObject(void) const {return object; }
    // mutators
    void setBlock(int _x, int _y, typeObject _object) {x = _x; y = _y; object = _object; }
};

class Sokoban : public sf::Drawable {
 public:
    // TODO(Daniel): develop moveplayer in part B
    void movePlayer(moveSet move);
    // muttator
    void setSokobanMatrix(vector<GameObject>& blocks);
    void setSprite(GameObject& block);
    // accessors
    // pre: passed x & y values inside bounds of matrix
    // post: returns GameObject at location
    GameObject& getGameObject(const int y, const int x) {return matrix[y][x]; }
    int getWidth(void) {return xSize; }
    int getHeight(void) {return ySize; }
    sf::Sprite getSprite(void) {return sprite; }
    friend ifstream& operator>>(ifstream& in, Sokoban& object);

 private:
    // overides sf::Drawable virtual function
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
    int xSize;
    int ySize;
    sf::Sprite sprite;
    sf::Texture texture;
    vector<vector<GameObject>> matrix;
};

// pre: is passed valid file stream & Sokoban object
// post: returns file stream & fills Sokoban
ifstream& operator>>(ifstream& in, Sokoban& object);

// insert is a helper fucntion for the >> operator overload
// pre: takes a vaild filestream, vector of gameObjects and coordintes
// post: returns true if item was succesfully put in vector
bool insert(ifstream& in, vector<GameObject>& objects, int x, int y);

// pre: Is passed valid Vector2f, GameObjects, and int pair, pixel sizes
// post: returns updated an updated Vector2f
void setDrawPos(sf::Vector2f& location, const GameObject& block, const pair<int, int>& xyCor);

```

Sokoban.cpp

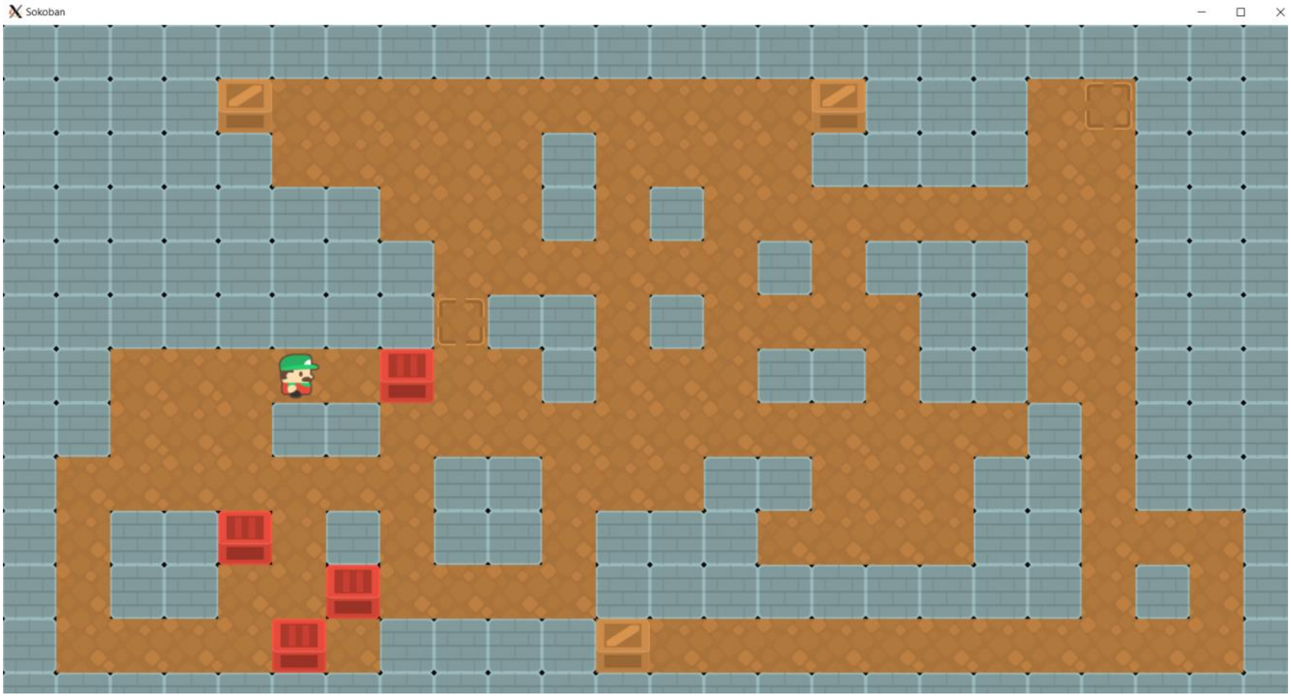
```
/* "Copyright [2023] <Daniel Bergeron>" */
#include "Sokoban.hpp"

GameObject::GameObject(int _x, int _y, typeObject _object) {
    x = _x;
    y = _y;
    object = _object;
}

void Sokoban::movePlayer(moveSet) {
    // TODO(Daniel): Finish.
    cout << "Player Doesn't feel like moving!\n";
    return;
}

void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    target.draw(sprite, states);
}

void Sokoban::setSprite(GameObject& block) {
// pulls textures from files based of block type
    switch (block.getTypeObject()) {
        case player:
            texture.loadFromFile("sokoban/images/player_05.png");
            break;
        case wall:
            texture.loadFromFile("sokoban/images/block_06.png");
            break;
        case floor:
            texture.loadFromFile("sokoban/images/ground_01.png");
            break;
        case crate:
            texture.loadFromFile("sokoban/images/crate_03.png");
            break;
        case storage_space:
            texture.loadFromFile("sokoban/images/ground_04.png");
            break;
        default:
            // Will set any other GameObject to wall
            // TODO(Daniel): Make code to deal with states that
            // are stored in matirx
            texture.loadFromFile("sokoban/images/block_06.png");
    }
    sprite.setTexture(texture);
}

ifstream& operator>>(ifstream& in, Sokoban& level) {
    // objects and var
    int xtemp, ytemp;
    vector<GameObject> objects, tempVec;
    // make sure matirx is clear before it is used again.
    level.matrix.clear();
    // take map size input
    in >> xtemp >> ytemp;
    if (xtemp <= 0 && ytemp <= 0) {
        exit(1);
    }
    level.xSize = xtemp;
    level.ySize = ytemp;

    // make matrix outline
    level.matrix.resize(ytemp);
    for (int i = 0; i < static_cast<int>(level.matrix.size()); i++) {
        for ( int j = 0; j < xtemp; j++ ) {
            GameObject empty;
            level.matrix[i].push_back(empty);
        }
    }

    // start reading chars
    // use to build Game objects
    // place in input vector then build matrix
    if (insert(in, objects, xtemp, ytemp)) {
        level.setSokobanMatrix(objects);
    }
    return in;
}

bool insert(ifstream& in, vector<GameObject>& objects, int xLimit, int yLimit ) {
    char input;
    int x = 0, y = 0;
    // Takes input and puts into a vector
    // based of what the char input is.
    // Two guards at the end to ensure
    // it dosen't go out of bounds.

    while ( !(in.eof()) ) {
        GameObject block;
        in >> input;
        switch ( input ) {
        case '@':
            block.setBlock(x, y, player);
            break;
        case '#':
            block.setBlock(x, y, wall);
            break;
        case '.':
            block.setBlock(x, y, floor);
```

```
         break;
            case 'A':
                block.setBlock(x, y, crate);
                break;
            case 'a':
                block.setBlock(x, y, storage_space);
                break;
            case '1':
                block.setBlock(x, y, stored);
                break;
            default:
                return false; }
            x++;
            if (y < yLimit) {
                objects.push_back(block);
            }
            if (x >= xLimit) {
                y++;
                x = 0;
            }
        }
        return true;
    }

    void Sokoban::setSokobanMatrix(vector<GameObject>& blocks) {
        // puts block in matrix based of the objects
        // predetermined location
        int x = 0, y = 0;
        int val = blocks.size();
        for (int i = 0; i < val; i++) {
            x = blocks[i].getXlocation();
            y = blocks[i].getYlocation();
            matrix[y][x] = blocks[i];
        }
    }

    void setDrawPos(sf::Vector2f& location, const GameObject& block, const pair<int, int>& xyCor) {
        location.x = block.getXlocation() * xyCor.first;
        location.y = block.getYlocation() * xyCor.second;
    }


```

PS2B – Sokoban

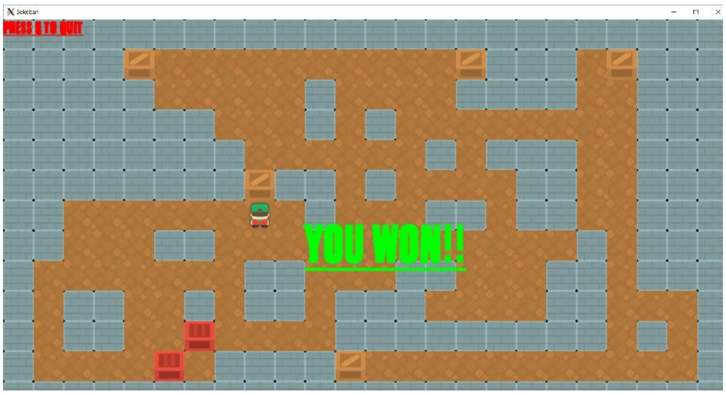
Program Discussion

In this program I built of the features I had in PS2A. This program implemented player movement, object movement and win conditions. This program is the first time I Had to manipulate visuals in this way. I took basic PNG images and brought them to life. This program required me to take a player PNG image and link it with the keyboard so when the play presses a key the image would respond and move in that direction. Then I built a system that would allow my player image to interact with the environment. I need rules, what blocks can the player move onto, what blocks can the player not move past, and what blocks can the player interact with. The player needs to move freely across the floor blocks. It needed to be restricted by the wall blocks. The player needed to be able to move the crate block, and finally change the crate block's state when it reaches a storage block. To accomplish this, I maintained my gameObject blocks that held that data. I then had the matrix for movement and whenever a move command was inputted it directly manipulates the GameObjects in the matrix. So, pressing the 'w' key activated the move function in my class which moved the block to the next point in the matrix. However as stated earlier the design oversight I made in the previous assignment came into play. When a player block moved it treated a floor block as something dynamic, hence the program did a few weird things. First every time the player moved the block underneath them state changed. This caused floor blocks to transform into other things. Sometimes it would just disappear. To fix this, I was forced to complicate my classes. More specifically my move function, which needed to have an added level of complexity to handle the niche bugs that kept appearing. The problem became amplified when the player interacted with crate blocks. Now my program had two blocks that were changing the states of other blocks around them. To combat this spiraling issue, I had to add a new class



Window Output 4

that handled the transition. I also had to add complicated conditionals to check the matrix for these potential situations. This whole situation that took me hours to fix could have been avoided if I had only made a static background. Essentially what I did by not including the static background was give the player the ability to manipulate the background. Through hours of tedious effort, I produced the `setMoveObjects` and the `executeMoveObjects` functions which resolved this issue. Though it should be noted that to accomplish this, the functions contain copious amounts of spaghetti code. If I ever redo this project this will be the first edit, I make. Other than this glaring issue there were some fun SFML feature I got to work with, I learned how to display text in my window. I was able to download a font and use it in my game. Learning how to use resources like fonts, sound bites and images was one of the more enjoyable parts of this project. Other parts of the project that I implemented successfully was the win conditions, player directional movement and finally the ability to restart the game at any point. For the win condition I created a function called `isWon`. This function went through the matrix and checked the state of the map. If all the crates had a stored state, the endgame loop triggered. I was able to accomplish the extra credit, that asked for the players orientation to change based on the direction of movement. I also implemented a feature that allowed a player to press a restart key, and the entire game would be set back to its original state. There where two extra credit options that I wanted to accomplish but could not be due to time and other issues. The first is I wanted to be able to play a sound when the player one. I had the sound bite implemented, but I had compatibility issues with my computer drivers and the SFML library that I could not resolve. The undo feature I couldn't implement due to time. However, if I had the chance to implement it, I would have created a stack object and would have pushed the state of the game onto the stack after every turn, and when the undo button was pressed, I would set the current state to the top of the stack then pop it. The final aspect of this program I felt I exceled at was implementing a texture map. Texture objects are heavy so creating new texture every time a sprite needs it heavily impacts program performance. To avoid this, I put all my textures in a map, thus enabling me to create multiple sprites that use the same object. I managed to accomplish everything I set out to do with this assignment and more. I learned many good lessons from this assignment, some the hard way.



Window Output 5

Key Elements Discussion

This program forced me to deal with OOP up close. I feel that while I was able to use some important object-oriented programing principles, I feel I fell short. My main class in this program is very heavy and I should have broken it down into smaller classes. My Sokoban class can only be used I am very specific way and doesn't feel modular. Yet despite this, my Sokoban program has improved my understanding extensively. In this program I also implemented an algorithm from the C++ algorithm library, I used the for_range function with Lambda. The lambda Is what checks the win condition, and it is applied over the entire matrix by the for_range function.

What I Learned

The important lesson I learned from this code was how bad design decisions can have consequences that are not always apparent until later, where they can become big problems. I learned it’s always better to pay the inconvenience tax in the moment to make your code more robust, rather than deal with it later.

Issues

There was one noticeable but small bug in my final program. If the player spams the reset button the player object bounces around the map. I only have theories for why this happens, but it seems the operation of reading the map into the Sokoban class is a heavy operation, and the spamming the restart button might change some data in the input buffer. I did not have time to really debug this, and it was an issue that did not affect the game at all. The only way the issue became present is if you were actively seeking to cause it, an average player would not run into it.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = Sokoban.o main.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system
DEPS = ps2b/Sokoban.hpp

all: Sokoban lint

Sokoban: $(OBJ)
	$(CC) $(CFLAGS) -o Sokoban $(OBJ) $(LIBS)
lint:
	cpplint *.cpp *.hpp
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) Sokoban lint

```

Main.cpp

```
/* "Copyright [2023] <Daniel Bergeron>" */
#include <string>
#include "ps2b/Sokoban.hpp"

moveSet getKeyBoardInput(void);
bool resetLevel(moveSet reKey, ifstream& fp, Sokoban& lv, std::string gameMap);
```

```cpp

int main(int argc, char* argv[]) {
    // Simple driver to load and test maps
    Sokoban lv;
    ifstream fp;
    moveSet input = noMove;
    std::string gameMap;
    int fontSize = 100;
    gameMap = argv[1];
    fp.open(gameMap);
    if (fp.fail()) {
        cout << "Failed to open file\n";
        return -1;
    }
    fp >> lv;
    fp.close();

    sf::Text text, text2;
    sf::Font font;
    if (!font.loadFromFile("endGameFont.ttf")) {
        cout << "Font failed to load\n";
    }
    text.setFont(font);
    text2.setFont(font);
    text.setString("YOU WON!!");
    text2.setString("Press Q to quit");
    text.setCharacterSize(100);
    text2.setCharacterSize(30);
    text.setPosition((lv.getTextureSizeX() * lv.getWidth())/2.50,
                 (lv.getTextureSizeY() * lv.getHeight())/2.00);
    text.setFillColor(sf::Color::Green);
    text2.setFillColor(sf::Color::Red);
    text.setStyle(sf::Text::Bold | sf::Text::Underlined);
    text2.setStyle(sf::Text::Bold | sf::Text::Underlined);

    auto endGame = [](moveSet in) {  // Lambda expression
        if (in == quiet) {
            return true;
        }
        return false;
    };

    sf::RenderWindow window1(sf::VideoMode(lv.getTextureSizeX() * lv.getWidth(),
    lv.getTextureSizeY() * lv.getHeight()), "Sokoban");

    while (window1.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window1.close();
            }
        }
        input = getKeyBoardInput();
        if (endGame(input)) {
            return 0;
        }
        if ( resetLevel(input, fp, lv, gameMap) ) {
            window1.clear(sf::Color::Black);
            input = getKeyBoardInput();
        }
        lv.movePlayer(input);
        window1.draw(lv);
        window1.display();
        while (lv.isWon() == true) {
            // inside the main loop, between window.clear()
            // and window.display()
            window1.draw(text);
            window1.draw(text2);
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
                return 0;
            }
            input = getKeyBoardInput();
            if (resetLevel(input, fp, lv, gameMap)) {
                window1.clear(sf::Color::Black);
            }
            window1.display();
            while (window1.pollEvent(event)) {
                if (event.type == sf::Event::Closed) {
                    window1.close();
                    return 0;
                }
            }
        }
    }
    return 0;
}

bool resetLevel(moveSet reKey, ifstream& fp, Sokoban& lv, std::string gameMap) {
        Sokoban newlv;
        if (reKey == restart) {
            fp.open(gameMap);
            fp >> lv;
            fp.close();
                  return true;
              }
              return false;
      }

      moveSet getKeyBoardInput(void) {
          if (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {
              while (sf::Keyboard::isKeyPressed(sf::Keyboard::W)) {}
              return up;
          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {
              while (sf::Keyboard::isKeyPressed(sf::Keyboard::S)) {}
              return down;
          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {
              while (sf::Keyboard::isKeyPressed(sf::Keyboard::A)) {}
              return left;
          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {
              while (sf::Keyboard::isKeyPressed(sf::Keyboard::D)) {}
              return right;
          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
              while (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {}
              return restart;
          } else if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
              return quiet;
          } else {
              return noMove;
          }
      }


```

Sokoban.hpp

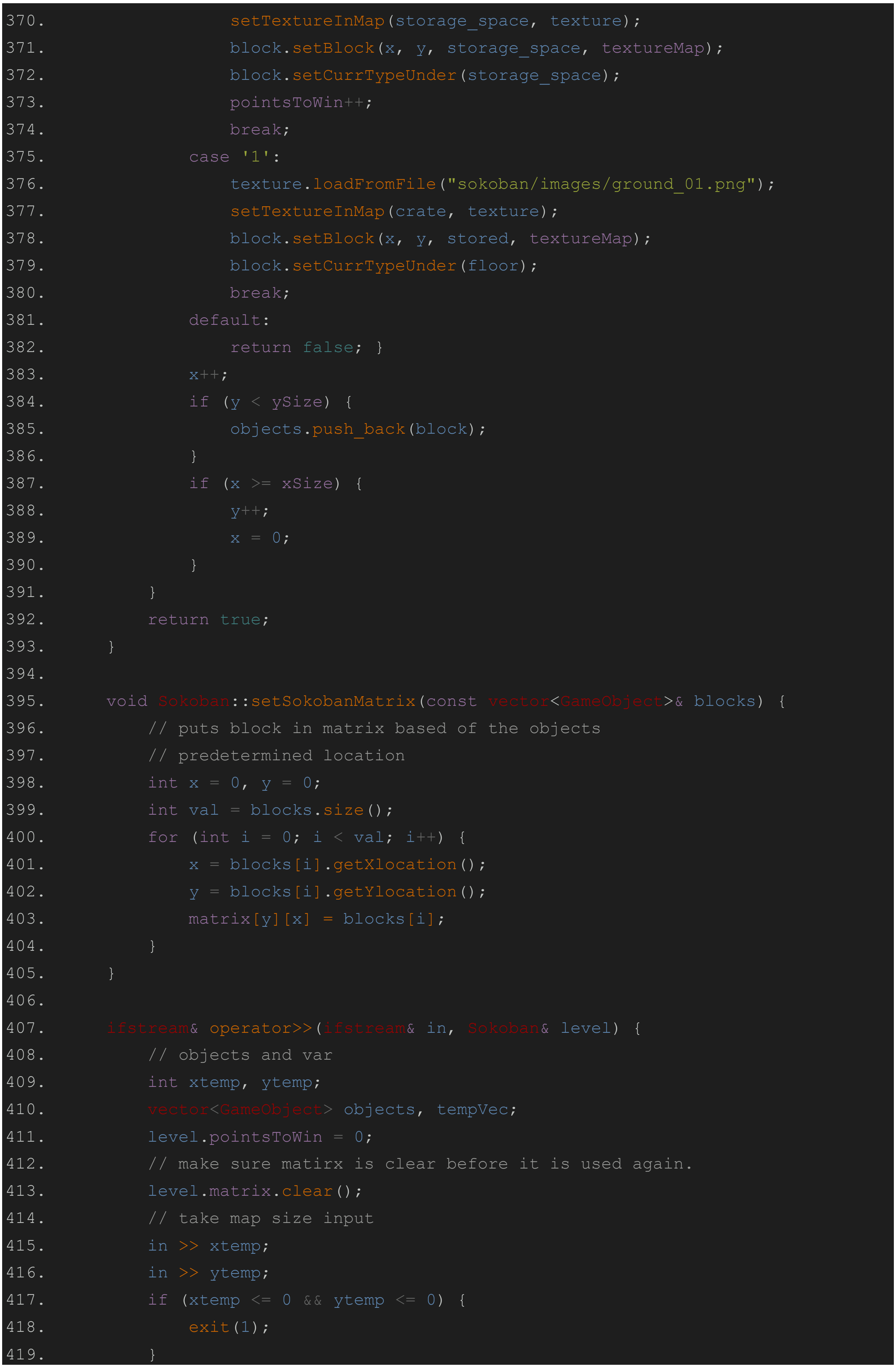
```
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS2B_PS2B_SOKOBAN_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS2B_PS2B_SOKOBAN_HPP_
/* "Copyright [2023] <Daniel Bergeron>" */
#include <fstream>
using std::ifstream;
#include <utility>
using std::pair;
#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <vector>
using std::vector;
#include <map>
using std::map;
#include <algorithm>
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>
#include <SFML/Audio.hpp>

// enum describes set of object types and move types
enum typeObject{nill, player, playerDown, playerUp,
playerLeft, playerRight, wall, floor, crate, storage_space, stored};
enum moveSet{noMove, up, down, left, right, restart, quiet};

```

```
class MoveObject {
 public:
      // constructor
      MoveObject();
      // accessor
      pair<int, int> getCurrSpace(void)const {return currSpace; }
      pair<int, int> getNextSpace(void)const {return nextSpace; }
      bool getMovable(void) {return moveable; }
      // mutators
      void setCurrSpace(pair<int, int> _next) {currSpace = _next; }
      void setNextSpace(pair<int, int> _next) {nextSpace = _next; }
      void setMoveable(bool _moveable) {moveable = _moveable; }
 private:
      bool moveable;
      pair<int, int> currSpace;
      pair<int, int> nextSpace;
};

class GameObject {
 private:
    int x;
    int y;
    typeObject object;
    typeObject currTypeUnder;  // the type under if object is layered
    sf::Sprite sprite;
 public:
    // constructors
    GameObject() : x(0), y(0), object(nill), currTypeUnder(nill) {}
    GameObject(int _x, int _y, typeObject _object);
    // accessors
    int getXlocation(void) const {return x; }
    int getYlocation(void) const {return y; }
    typeObject getCurrTypeUnder(void) const {return currTypeUnder; }
    typeObject getTypeObject(void) const {return object; }
    sf::Sprite getSprite(void) const {return sprite; }
    // mutators
    void setBlock(int _x, int _y, typeObject _object,
                              map<typeObject, sf::Texture>& texture);
    void setCurrTypeUnder(typeObject type) {currTypeUnder = type; }
    void setSprite(sf::Sprite _sprite) {sprite = _sprite; }
};

class Sokoban : public sf::Drawable {
 public:
    // constructor
    Sokoban();
    // pre: is given a move command
    // post: player will be moved by command
    void movePlayer(moveSet move);
    // muttator
    void setSokobanMatrix(const vector<GameObject>& blocks);
    void setPlayerPosition(int x, int y) {playerLocation.first = x,
                                          playerLocation.second = y; }
    // accessors
    // pre: passed x & y values inside bounds of matrix
    // post: returns GameObject at location
    GameObject& getGameObject(const int y, const int x) {return matrix[y][x]; }

    typeObject getTypeUnderAtPosition(pair<int, int>);
    int getWidth(void) const {return xSize; }
    int getHeight(void) const  {return ySize; }
    int getTextureSizeX(void) const {return textureSize.first; }
    int getTextureSizeY(void) const {return textureSize.second; }
    GameObject& getNextSpace(moveSet move, pair<int, int> CurrObjectLoc);
    const bool isWon(void);
    friend ifstream& operator>>(ifstream& in, Sokoban& object);

 private:
    // overides sf::Drawable virtual function
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
    // is a helper fucntion for the >> operator overload
    // pre: takes a vaild filestream, vector of gameObjects and coordintes
    // post: returns true if item was succesfully put in vector
          bool buildGameObjectVector(ifstream& in, vector<GameObject>& objects);
          // pre: takes valid type and texture reffrence
          // post: sets inside texture map
          void setTextureInMap(typeObject type, sf::Texture& texture);
          // pre: takes commands
          // post: returns list of commands in order
          void setMoveObjects(moveSet move);
          // pre: is called
          // updates matrix using movevlocks
          bool executeMoveObjects(moveSet move);
          // pre: is given two valid move blocks
          // post: places them in matrix location
          void moveMatrixBlocks(const GameObject& replaceBlock,
                                    const GameObject& movingBlock);
          // pre: is given a move
          // post: returns enum for animation type
          typeObject getPlayerMovement(moveSet move);

          int xSize;
          int ySize;
          int pointsToWin;
          typeObject lastMove = player;
          MoveObject nextPlayerMove;
          MoveObject nextBlockMove;
          pair<int, int> playerLocation;
          pair<int, int> currentCrateLocation;
          pair<int, int> textureSize;
          vector<vector<GameObject>> matrix;
          map<typeObject, sf::Texture> textureMap;
      };

      // pre: is passed valid file stream & Sokoban object
      // post: returns file stream & fills Sokoban
      ifstream& operator>>(ifstream& in, Sokoban& object);

      // pre: Is passed valid Vector2f, GameObjects, and int pair, pixel sizes
      // post: returns updated an updated Vector2f
      void setDrawPos(sf::Vector2f& location, const GameObject& block,
                        const pair<int, int>& xyCor);

      // pre: passed a valid texture size
      // post: returns a pair with xy size
      void getTexturePixelSize(const sf::Texture&, pair<int, int>& cor);

      #endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS2B_PS2B_SOKOBAN_HPP_

```

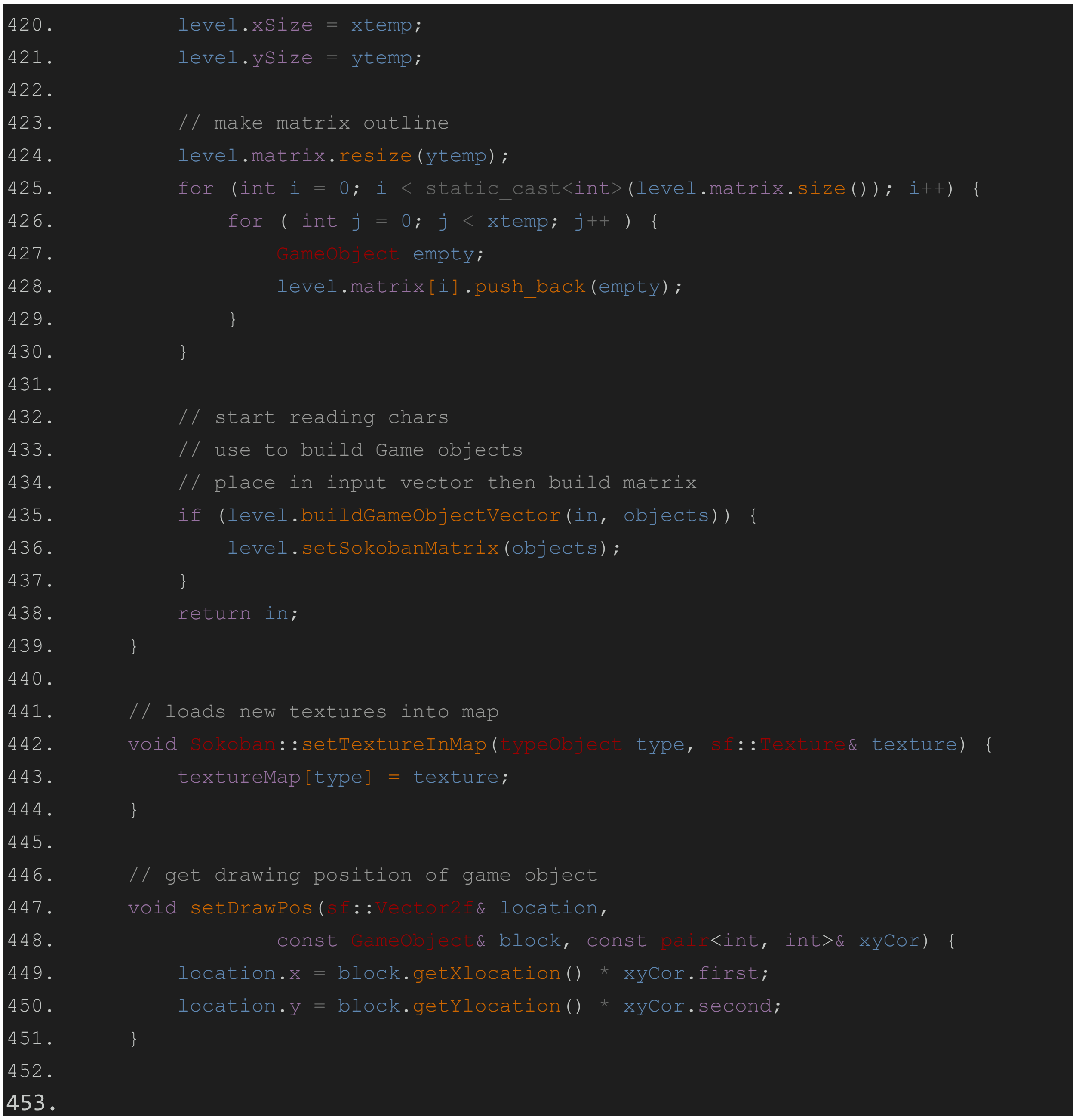
Sokoban.cpp

```
/* "Copyright [2023] <Daniel Bergeron>" */
#include "ps2b/Sokoban.hpp"

GameObject::GameObject(int _x, int _y, typeObject _object) {
    x = _x;
    y = _y;
    object = _object;
    currTypeUnder = _object;
}

void GameObject::setBlock(int _x, int _y, typeObject _object,
                            map<typeObject, sf::Texture>& texture) {
    // reasigns textures
    x = _x;
    y = _y;
    object = _object;
    // gets texture from map of stored textures
    // so no new textures are created.
    sprite.setTexture(texture[_object]);
    if (_object == playerDown || _object == playerLeft
        || _object == playerRight || _object == playerUp) {
            object = player;
    }
}

MoveObject::MoveObject() {
    pair<int, int> empty;
    empty.first = 0;
    empty.second = 0;
    currSpace = empty;
    nextSpace = empty;
    moveable = false;
}

Sokoban::Sokoban() {
    sf::Texture newText;
    xSize = 0;
    ySize = 0;
    newText.loadFromFile("sokoban/images/crate_02.png");
    getTexturePixelSize(newText, textureSize);
    textureMap[stored] = newText;
    pointsToWin = 0;
    playerLocation.first = 0;
    playerLocation.second = 0;
}

// gets the pixels sizes of the the texture passed in
void getTexturePixelSize(const sf::Texture& texture, pair<int, int>& cor) {
    sf::Vector2u pixels;
    pixels = texture.getSize();
    cor.first = static_cast<int>(pixels.x);
    cor.second = static_cast<int>(pixels.y);
    return;
}

// functions takes move
// checks for no move then
// sets Sokobans move Objects then
// it excuetes the commands within
// the move objects
void Sokoban::movePlayer(moveSet move) {
    MoveObject moveInfo;
    if (move == noMove) {
        return;
    }

    setMoveObjects(move);
    executeMoveObjects(move);
    return;
}

// returns the type under from anypoint in the matrix
// importent for setting new blocks
typeObject Sokoban::getTypeUnderAtPosition(pair<int, int> position) {
    return matrix[position.second][position.first].getCurrTypeUnder();
}

// checks win condition
// returns win/true when all
// points of ben aquired by player
const bool Sokoban::isWon(void) {
    // std::count_if(v.begin(), v.end(), [](int i) { return i % 4 == 0; });
    int points = 0;
    std::for_each(matrix.begin(), matrix.end(),
    [&](vector<GameObject> i) {
        for (GameObject obj : i) {
            if (obj.getTypeObject() == stored) {
                points++;
            }
        }
    });
    // cout << "pointed scored score: " << points << endl;
    if (pointsToWin > points) {
        return false;
    } else {
        return true;
    }
}

// function goal is to validate if the next space is avaible for movement
void Sokoban::setMoveObjects(moveSet move) {
    GameObject nextBlock, nextCrateBlock;
    pair<int, int> previousLoc, newLoc;
    // see if player can move
    // getSpace at move
    nextPlayerMove.setMoveable(false);
    nextBlockMove.setMoveable(false);
    nextBlock = getNextSpace(move, playerLocation);

    // no move if wall or object is in state stored
    if (nextBlock.getTypeObject() == wall
            || nextBlock.getTypeObject() == stored) {
        nextPlayerMove.setMoveable(false);
        return;
    // if next block is crate condition
    } else if (nextBlock.getTypeObject() == crate) {
        // checks to see if a crate is a moveable object
        currentCrateLocation.first = nextBlock.getXlocation();
        currentCrateLocation.second = nextBlock.getYlocation();
        nextCrateBlock = getNextSpace(move, currentCrateLocation);
        // if wall or another crate both player and crate can't move
        if (nextCrateBlock.getTypeObject() == wall
                || nextCrateBlock.getTypeObject() == crate) {
            nextBlockMove.setMoveable(false);
            nextPlayerMove.setMoveable(false);
        } else if (nextCrateBlock.getTypeObject() == storage_space
                    || nextCrateBlock.getTypeObject() == floor) {
            // set move player
            nextPlayerMove.setMoveable(true);
```

```
                nextPlayerMove.setCurrSpace(playerLocation);
                newLoc.first = nextBlock.getXlocation();
                newLoc.second = nextBlock.getYlocation();
                nextPlayerMove.setNextSpace(newLoc);

                // set move block
                nextBlockMove.setMoveable(true);
                previousLoc.first = nextBlock.getXlocation();
                previousLoc.second = nextBlock.getYlocation();
                nextBlockMove.setCurrSpace(previousLoc);
                newLoc.first = nextCrateBlock.getXlocation();
                newLoc.second = nextCrateBlock.getYlocation();
                nextBlockMove.setNextSpace(newLoc);
            }
            return;
        }
        // default condtion player is movable
        nextPlayerMove.setMoveable(true);
        nextPlayerMove.setCurrSpace(playerLocation);
        newLoc.first = nextBlock.getXlocation();
        newLoc.second = nextBlock.getYlocation();
        nextPlayerMove.setNextSpace(newLoc);
    }

    // checks what game object is at the next space based of user input
    GameObject& Sokoban::getNextSpace(moveSet move, pair<int, int> CurrObjectLoc) {
        int nextX, nextY;
        nextX = CurrObjectLoc.first;
        nextY = CurrObjectLoc.second;

        switch (move) {
        case up:
            nextY -= 1;
            break;
        case down:
            nextY += 1;
            break;
        case left:
            nextX -= 1;
            break;
        case right:
            nextX += 1;
            break;
        default:
            return matrix[CurrObjectLoc.first][CurrObjectLoc.second];
            break;
        }
        return matrix[nextY][nextX];
    }

    // function uses two moveObjects one to replace
    // the object on the space it is being moved from
    // i.e. the replace object, the other object is
    // the one being move to another location the moving
    // object, which retains its curr type but the type
    // under changes to that of the object which was
    // originaly there
    bool Sokoban::executeMoveObjects(moveSet move) {
        GameObject replaceBlock, movingBlock;
        typeObject playerMovement = player;
        pair<int, int> oldPosition, newPosition;
        if (nextPlayerMove.getMovable() == false
                && nextBlockMove.getMovable() == false) {
            return false;
        } else if (nextPlayerMove.getMovable() == true
                    && nextBlockMove.getMovable() == true) {
            // move crate & updateMap;
            oldPosition = nextBlockMove.getCurrSpace();
            newPosition = nextBlockMove.getNextSpace();
            // if crate is on storage space change to stored state
            if (getTypeUnderAtPosition(newPosition) == storage_space) {
                replaceBlock.setBlock(oldPosition.first, oldPosition.second,
                            getTypeUnderAtPosition(oldPosition), textureMap);
                // set stored state on crate at new location
                movingBlock.setBlock(newPosition.first,
                            newPosition.second, stored, textureMap);
                movingBlock.setCurrTypeUnder(getTypeUnderAtPosition(newPosition));
                moveMatrixBlocks(replaceBlock, movingBlock);
            } else {
                replaceBlock.setBlock(oldPosition.first, oldPosition.second,
                        getTypeUnderAtPosition(oldPosition), textureMap);
                replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());

                movingBlock.setBlock(newPosition.first,
                        newPosition.second, crate, textureMap);
                movingBlock.setCurrTypeUnder(getTypeUnderAtPosition(newPosition));
                moveMatrixBlocks(replaceBlock, movingBlock);
            }
            // move player, updateMap, update currLocation
            oldPosition = nextPlayerMove.getCurrSpace();
            newPosition = nextPlayerMove.getNextSpace();

            replaceBlock.setBlock(oldPosition.first, oldPosition.second,
                        getTypeUnderAtPosition(oldPosition), textureMap);
            replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());
            // moving block is player
            playerMovement = getPlayerMovement(move);
            movingBlock.setBlock(newPosition.first,
                        newPosition.second, playerMovement, textureMap);
            movingBlock.setCurrTypeUnder(
                    matrix[newPosition.second][newPosition.first].getTypeObject());
            playerLocation = newPosition;
            moveMatrixBlocks(replaceBlock, movingBlock);
            return true;
        } else if (nextPlayerMove.getMovable() == true
                        && nextBlockMove.getMovable() == false) {
            oldPosition = nextPlayerMove.getCurrSpace();
            newPosition = nextPlayerMove.getNextSpace();

            // always get the type that will be under player
            replaceBlock.setBlock(oldPosition.first, oldPosition.second,
                        getTypeUnderAtPosition(oldPosition), textureMap);
            replaceBlock.setCurrTypeUnder(replaceBlock.getTypeObject());

            //  player
            playerMovement = getPlayerMovement(move);
            movingBlock.setBlock(newPosition.first,
                        newPosition.second, playerMovement, textureMap);
            movingBlock.setCurrTypeUnder(
                    matrix[newPosition.second][newPosition.first].getTypeObject());
            playerLocation = newPosition;
            moveMatrixBlocks(replaceBlock, movingBlock);
            return true;
        }
        return false;
    }

    typeObject Sokoban::getPlayerMovement(moveSet move) {
        switch (move) {
        case up:
            return playerUp;
        case down:
            if (lastMove == player) {
                lastMove = playerDown;
                return playerDown;
            } else {
                lastMove = player;
                return player;
            }
            return playerDown;
        case left:
            return playerLeft;
        case right:
            return playerRight;
        default:
            return player;
        }
    }

    // replaces the block being moved with replace block
    // and puts moving block in its new position
    void Sokoban::moveMatrixBlocks(const GameObject& replaceBlock,
                                        const GameObject& movingBlock) {
        matrix[replaceBlock.getYlocation()][replaceBlock.getXlocation()]
                    = replaceBlock;
        matrix[movingBlock.getYlocation()][movingBlock.getXlocation()]
                    = movingBlock;
    }

    // renders the game map using the information
    // contained in the matrix's gameObjects
    void Sokoban::draw(sf::RenderTarget& target, sf::RenderStates states) const {
        GameObject block;
        sf::Vector2f location;
        sf::Texture floorTexture;
        sf::Sprite floorSprite, sprite;
        map<typeObject, sf::Texture>::const_iterator pos = textureMap.find(floor);
        floorTexture = pos->second;
        floorSprite.setTexture(floorTexture);
        target.draw(sprite, states);
        // pulls blocks from matrix to be drawn
        for (int y = 0; y < ySize; y++) {
            for (int x = 0; x < xSize; x++) {
                block = matrix[y][x];
                setDrawPos(location, block, textureSize);
                // if there is a layered block the type under must be drawn first
                if (block.getTypeObject() == player
                        || block.getTypeObject() == stored) {
                    pos = textureMap.find(block.getCurrTypeUnder());
                    floorTexture = pos->second;
                    floorSprite.setTexture(floorTexture);
                    floorSprite.setPosition(location);
                    target.draw(floorSprite, states);
                }
                sprite = matrix[y][x].getSprite();
                sprite.setPosition(location);
                target.draw(sprite, states);
            }
        }
    }

    bool Sokoban::buildGameObjectVector(ifstream& in, vector<GameObject>& objects) {
        char input;
        int x = 0, y = 0;
        // Takes input and puts into a vector
        // based of what the char input is.
        // Two guards at the end to ensure
        // it dosen't go out of bounds.

        while ( !(in.eof()) ) {
            GameObject block;
            sf::Texture texture;
            in >> input;
            switch ( input ) {
            case '@':
                texture.loadFromFile("sokoban/images/player_up.png");
                setTextureInMap(playerUp, texture);
                texture.loadFromFile("sokoban/images/player_down.png");
                setTextureInMap(playerDown, texture);
                texture.loadFromFile("sokoban/images/player_left.png");
                setTextureInMap(playerLeft, texture);
                texture.loadFromFile("sokoban/images/player_right.png");
                setTextureInMap(playerRight, texture);
                texture.loadFromFile("sokoban/images/player_05.png");
                setTextureInMap(player, texture);
                block.setBlock(x, y, player, textureMap);
                playerLocation.first = x;
                playerLocation.second = y;
                block.setCurrTypeUnder(floor);
                break;
            case '#':
                texture.loadFromFile("sokoban/images/block_06.png");
                setTextureInMap(wall, texture);
                block.setBlock(x, y, wall, textureMap);
                block.setCurrTypeUnder(floor);
                break;
            case '.':
                texture.loadFromFile("sokoban/images/ground_01.png");
                setTextureInMap(floor, texture);
                block.setBlock(x, y, floor, textureMap);
                block.setCurrTypeUnder(floor);
                break;
            case 'A':
                texture.loadFromFile("sokoban/images/crate_03.png");
                setTextureInMap(crate, texture);
                block.setBlock(x, y, crate, textureMap);
                block.setCurrTypeUnder(floor);
                break;
            case 'a':
                texture.loadFromFile("sokoban/images/ground_04.png");
```

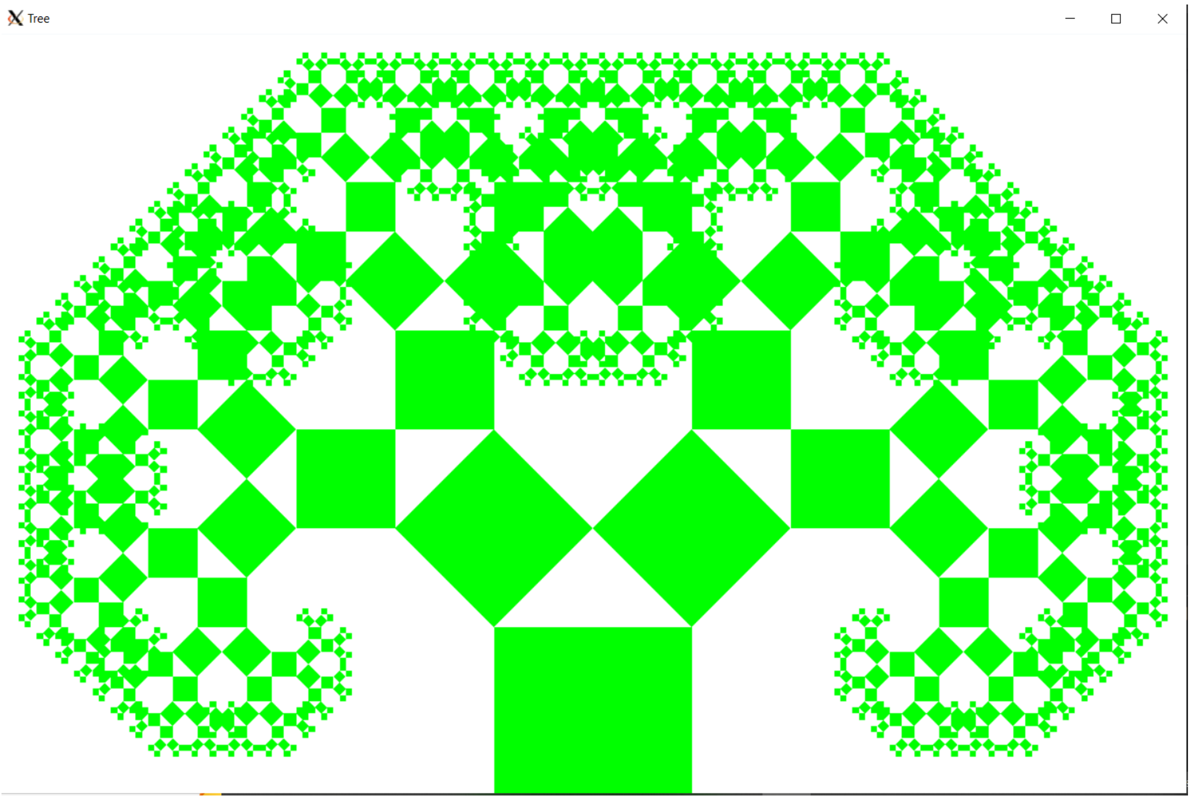


```cpp
            setTextureInMap(storage_space, texture);
            block.setBlock(x, y, storage_space, textureMap);
            block.setCurrTypeUnder(storage_space);
            pointsToWin++;
            break;
        case '1':
            texture.loadFromFile("sokoban/images/ground_01.png");
            setTextureInMap(crate, texture);
            block.setBlock(x, y, stored, textureMap);
            block.setCurrTypeUnder(floor);
            break;
        default:
            return false; }
        x++;
        if (y < ySize) {
            objects.push_back(block);
        }
        if (x >= xSize) {
            y++;
            x = 0;
        }
    }
    return true;
}

void Sokoban::setSokobanMatrix(const vector<GameObject>& blocks) {
    // puts block in matrix based of the objects
    // predetermined location
    int x = 0, y = 0;
    int val = blocks.size();
    for (int i = 0; i < val; i++) {
        x = blocks[i].getXlocation();
        y = blocks[i].getYlocation();
        matrix[y][x] = blocks[i];
    }
}

ifstream& operator>>(ifstream& in, Sokoban& level) {
    // objects and var
    int xtemp, ytemp;
    vector<GameObject> objects, tempVec;
    level.pointsToWin = 0;
    // make sure matirx is clear before it is used again.
    level.matrix.clear();
    // take map size input
    in >> xtemp;
    in >> ytemp;
    if (xtemp <= 0 && ytemp <= 0) {
        exit(1);
    }
```



```
        level.xSize = xtemp;
        level.ySize = ytemp;

        // make matrix outline
        level.matrix.resize(ytemp);
        for (int i = 0; i < static_cast<int>(level.matrix.size()); i++) {
            for ( int j = 0; j < xtemp; j++ ) {
                GameObject empty;
                level.matrix[i].push_back(empty);
            }
        }

        // start reading chars
        // use to build Game objects
        // place in input vector then build matrix
        if (level.buildGameObjectVector(in, objects)) {
            level.setSokobanMatrix(objects);
        }
        return in;
    }

    // loads new textures into map
    void Sokoban::setTextureInMap(typeObject type, sf::Texture& texture) {
        textureMap[type] = texture;
    }

    // get drawing position of game object
    void setDrawPos(sf::Vector2f& location,
                const GameObject& block, const pair<int, int>& xyCor) {
        location.x = block.getXlocation() * xyCor.first;
        location.y = block.getYlocation() * xyCor.second;
    }


```

PS3 – Pythagoras Tree



Program Discussion

This assignment required me to implement a Pythagorean Tree. The idea was to create a class called PTree that uses a recursive function to draw fractal in the shape of a tree. I was able to accomplish this by using the visual capabilities of the SFML library and a vector data structure. On top of this, to do the math required by this programming I had to use the Cmath library. This is the first time in a C++ program I was required to use a C library. It added an interesting dynamic to my linter, since the linter requires the header file to be ordered in a specific way. This program did not require much in the way of programing, the logic behind it made it difficult. This program forced me to really think about recursion. Recursions can be difficult to manage, but I was able to break it down. I used a recursion to create squares of different orientations and place them in a vector for the draw function to use later. I broke the recursion into two functions. The first function was not recursive. It was responsible for the base case, a larger square that would act as the platform from which the other squares are derived. After the base case was handled the function called the recursive function. This recursive function made the next two squares and then called itself twice on the two new squares. This program also used the command line to get information about how to draw the shape. I created a class to store sf::rectangles. Though this class did not do much for this program, I wanted to create something modular and simple to interact with SFML in my place. In this program the SFML documentation was crucial. Once I figured out the logic, the only difficult part of this program was the draw function. Initially I was only using global bounds to interact with the sf::rectangle object.

This however refers to its location on my entire screen, and not its position in the window I generated. This caused the program shapes to draw in strange areas. The other problem in my function to calculate the size of a square is that I accidentally made a miss calculation that resulted in my square size being negative. This did not cause a crash but rather it inverted all the positional data in my square object causing them to draw in random places. Overall, a good assignment, that furthered my understanding of recursion.

Key Elements Discussion

Vectors were crucial for me since they allowed me to create all the visuals in the recursion and store them for later use. This made my draw function very simple since all it had to do was run through the vector and draw each object. I did not want to mix the visuals with the background work so created independent systems for each which were called in the driver together to accomplish the task of making the tree. Another important element of this code was the logic behind the recursive elements of the program. In each step the program uses the previous lengths to calculate the next size and trigonometry properties to calculate the angles of the new rectangles. This was done in two recursions. I used this to work down the left and write sub-tree. This project required me to develop an interesting algorithm and implement it using object-oriented programing. This project really forced me to think about algorithms in a different way. I feel it has grown my ability to understand and translate problems.

What I Learned

In this program I learned the true power of recursion, and how it can take a complex problem and trivialize it. If you looked at the output of this code alone, you would think this program is a very complex problem, but the recursion simplified the code greatly.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = PTree.o PTree_driver.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system
DEPS = PTree.hpp

all: PTree lint

PTree: $(OBJ)
          $(CC) $(CFLAGS) -o PTree $(OBJ) $(LIBS)
      lint:
          cpplint *.cpp *.hpp
      %.o: %.cpp $(DEPS)
          $(CC) $(CFLAGS) -c $<
      clean:
          rm -f $(OBJ) PTree lint

```

Main.cpp

```cpp
// Copyright [2023] <Daniel Bergeron>
#include "ps3_header/PTree.hpp"

int main(int argc, char* argv[]) {
    string lBase, nDepth;
    double L;
    int N;
    lBase = argv[1];
    nDepth = argv[2];
    L = stod(lBase);
    N = stoi(nDepth);
    cout << "The Base triangle size input is " << L << endl;
    cout << "The depth input is " << N << endl;
    PTree sappling(N, L);
    sappling.pTree();

    sf::RenderWindow window1(sf::VideoMode(6 * L, 4 * L), "Tree");

    while (window1.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                window1.close();
            }
        }
        window1.clear(sf::Color::White);
        window1.draw(sappling);
        window1.display();
    }
    return 0;
}


```

PTree.hpp

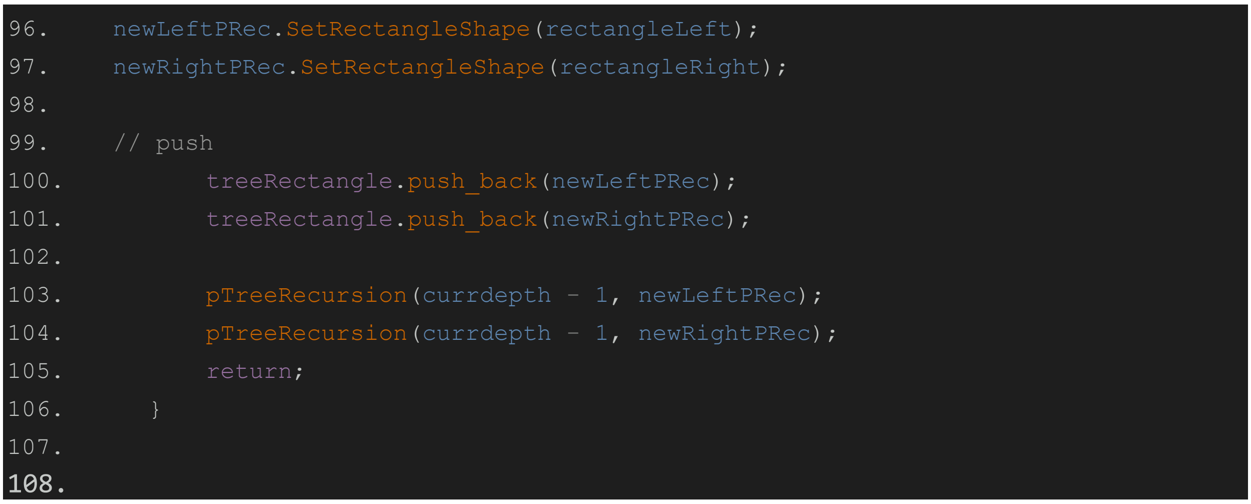
```cpp
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS3_PS3_HEADER_PTREE_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS3_PS3_HEADER_PTREE_HPP_
// Copyright [2023] <Daniel Bergeron>
// ptree class that derives from sf::Drawable
#include <iostream>
using std::cout;
using std::endl;
#include <utility>
using std::pair;
#include <string>
using std::string;
using std::stoi;
using std::stod;
#include <algorithm>
```

```
#include <vector>
using std::vector;
#include <cmath>
using std::sin;
using std::cos;
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

const float THETA = 45;

// need an object to hold rectangle
// hold a pair for its map loaction
// tells draw func where and how to draw
class PRec {
 public:
      // constructors
      PRec();
      PRec(sf::RectangleShape _rectangle, int _theta);
      // accessors
      sf::RectangleShape getRectangleShape(void) {return rectangle; }
      int getTheta(void) {return theta; }
      // muttators
      void SetRectangleShape(sf::RectangleShape _rectangle)
                                     {rectangle = _rectangle; }
      void setTheta(int _theta) {theta = _theta; }
 private:
      int theta;
      sf::RectangleShape rectangle;
};

class PTree : public sf::Drawable {
 public:
    PTree();
    PTree(int _depth, double _baseRecSize);

    // uses recurison to produce tree
    // pre: is called
    // post: tree is drawn
    void pTree(void);
 private:
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;
    void pTreeRecursion(int currdepth, PRec base);
    // helper function to ptree
    // this function makes a rectangle
    vector<PRec> treeRectangle;
    int depth;
    double baseRecSize;
};

```

```
#endif  //  _HOME_DBERGERO3_PROGRAMS_COMP4_PS3_PS3_HEADER_PTREE_HPP_


```

PTree.cpp

```
// Copyright [2023] <Daniel Bergeron>
#include "ps3_header/PTree.hpp"

PRec::PRec() {
    theta = 0;
    rectangle.setSize(sf::Vector2f(0, 0));
}
PRec::PRec(sf::RectangleShape _rectangle, int _theta) {
    rectangle = _rectangle;
    theta = _theta;
}

PTree::PTree() {
    depth = 0;
    baseRecSize = 0.0;
}
PTree::PTree(int _depth, double _baseRecSize) {
    depth = _depth;
    baseRecSize = _baseRecSize;
}

void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    sf::RectangleShape shape;
    PRec rectangle;
    int i;
    for (i = 0; i < treeRectangle.size(); i++) {
        rectangle = treeRectangle[i];
        shape = rectangle.getRectangleShape();
        shape.setFillColor(sf::Color::Green);
        target.draw(shape, states);
    }
    return;
}

void PTree::pTree(void) {
    if (depth <= 0 && baseRecSize <= 0) {
        return;
    }
    sf::RectangleShape baseShape(sf::Vector2f(static_cast<float>(baseRecSize),
                        static_cast<float>(baseRecSize)));
    PRec basePRec;
    pair<float, float> coord;
    coord.first = (baseRecSize * 3.0) - (baseShape.getSize().x/2.0);
    coord.second = (baseRecSize * 3.5) - (baseShape.getSize().x/2.0);
    baseShape.setPosition(coord.first, coord.second);
    basePRec.SetRectangleShape(baseShape);
    basePRec.setTheta(0);
    treeRectangle.push_back(basePRec);

    pTreeRecursion(depth, basePRec);
}

// function will make two rectangles and push them onto the vector
void PTree::pTreeRecursion(int currdepth, PRec base) {
    if (currdepth == 0) {
        return;
    }
    PRec newLeftPRec, newRightPRec;
    sf::RectangleShape rectangleLeft, rectangleRight;
    sf::Vector2f sizeL, sizeR;

    // set the size
    sizeL.x = (base.getRectangleShape().getSize().x * cos((M_PI/180) * THETA));
    sizeL.y = sizeL.x;
    cout << sizeL.x << endl;
    sizeR.x = (base.getRectangleShape().getSize().y * sin((M_PI/180) * THETA));
    sizeR.y = sizeR.x;

    rectangleLeft.setSize(sizeL);
    rectangleRight.setSize(sizeR);

    // make new theta
    newLeftPRec.setTheta(base.getTheta() - THETA);
    newRightPRec.setTheta(base.getTheta() + THETA);

    // positioning
    sf::Vector2f posL, posR;
    sf::Rect gridLoc = base.getRectangleShape().getLocalBounds();

    rectangleLeft.setOrigin(0, rectangleLeft.getSize().y);
    rectangleRight.setOrigin(rectangleRight.getLocalBounds().width,
                    rectangleRight.getLocalBounds().height);

    posL = base.getRectangleShape().getTransform().
            transformPoint(gridLoc.left, gridLoc.top);
    posR = base.getRectangleShape().getTransform().
            transformPoint(gridLoc.left + gridLoc.width, gridLoc.top);

    rectangleLeft.setPosition(posL);
    rectangleRight.setPosition(posR);
    // applying to rectangle object

    rectangleLeft.setRotation(newLeftPRec.getTheta());
    rectangleRight.setRotation(newRightPRec.getTheta());

```



```
    newLeftPRec.SetRectangleShape(rectangleLeft);
    newRightPRec.SetRectangleShape(rectangleRight);

    // push
           treeRectangle.push_back(newLeftPRec);
           treeRectangle.push_back(newRightPRec);

           pTreeRecursion(currdepth - 1, newLeftPRec);
           pTreeRecursion(currdepth - 1, newRightPRec);
           return;
       }


```

PS4A – Checkers

Program Discussion

This assignment involved setting up a checkers board for a checkers game. It did not require the addition of the actual game play mechanics, but it did require the development of an interactive highlight feature. Whenever a player moused over a piece the background becomes yellow. I was able to get all these features working properly. The key takes aways from this program for me was learning from past mistakes. In Sokoban I did not make the background static which caused me problems down the road. In this program I created the checkerboard/background in the constructor. After this I did not have to worry about losing visuals when the dynamic elements were introduced, since everything would be layered over this background. In the constructor I also loaded all the textures into a texture into a map. This allowed me to access them at my convenience at anytime in my other functions. In my constructor I loaded the pieces into my piece matrix. Which in my head was a separate entity. In the draw function the checkerboard and matrix would be brought together to generate the game time board. It was here that I made an interesting design choice. In the class Checkers there existed a matrix that contained an object from the piece class. It was these pieces that contained all the information the game required for movement and visuals. I treated my board as a matrix filled with light bulbs, where a light is either on or off, light is visible, or it is not visible. In the same vein the entire matrix was filled with pieces but somewhere *on* while somewhere *off.* This was so in future assignments when I needed to move a piece, I didn't actually move anything I simply turned one piece off and another on. This also played into the selection feature of a piece. A piece is either selected or it is not. When it was turned on it communicated to the draw function that it needed to draw a highlight before the piece. Since this is all contained to the pieces and by extension the matrix they exist in, it is impossible for the user to alter the integrity of the background. I feel that my development of this program grew Sokoban and better channeled Object-Oriented Programing principles. I did run into a new problem. The problem was a result of left over input from the mouse causing extra input to affect the program, this was resolved by creating a clearMouseBuffer function. A simple function that loops while the program picks up extra input data.

Key Elements Discussion

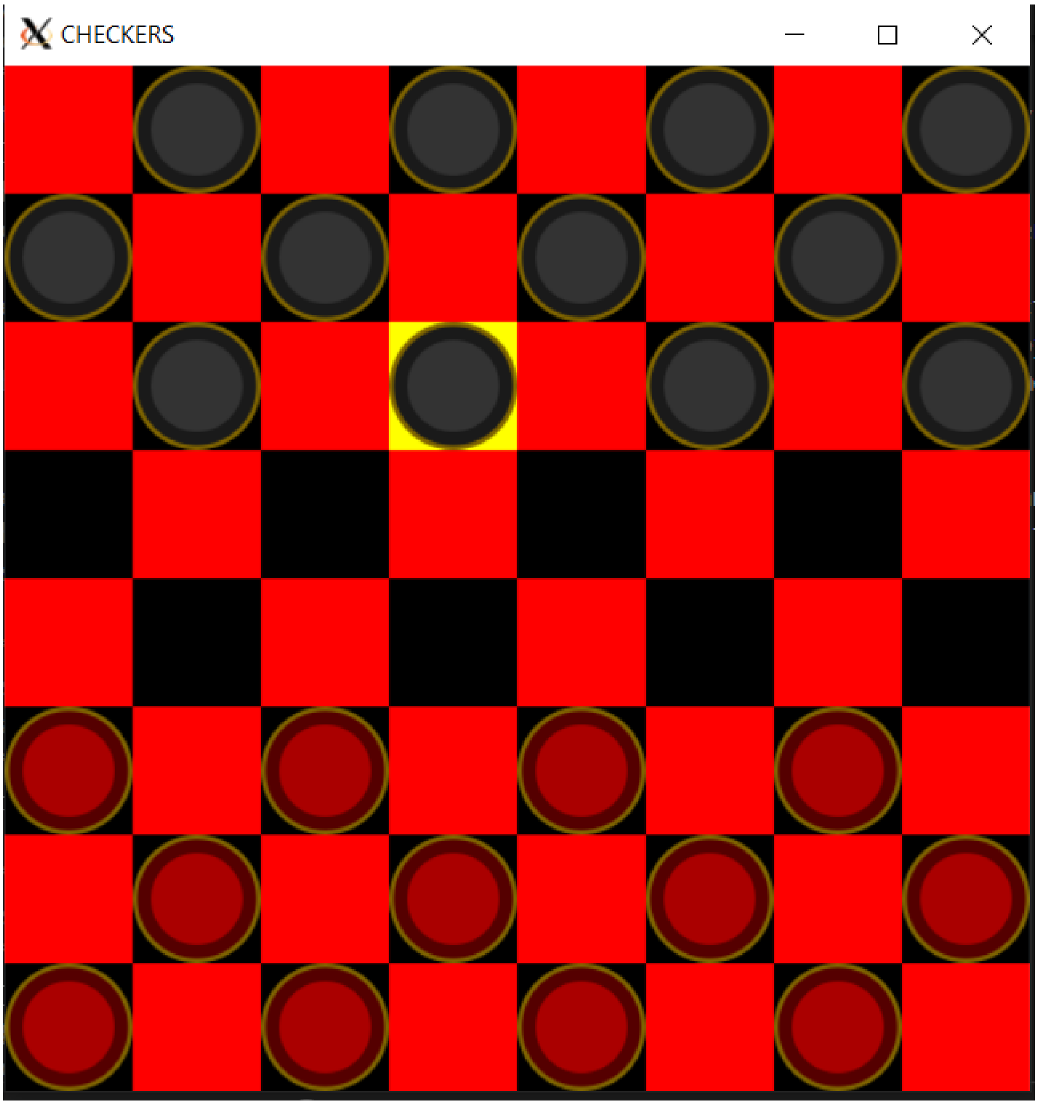
There are two elements that were central to this assignment, the first being the vector. I used a vector to create a matrix in which I placed and stored store game pieces which were crucial to this program. Pieces are a class used to give each piece characteristics that allow it to interact with the checkers in main as its own little interphase, that builds into the entire checkers' game. This program helped me to continue to develop my understanding of object-oriented programs. And, while I did not have to implement any complicated algorithms for this project, it did give me the opportunity to build off my failures in the last assignment.

What I Learned

In this program I learned how to go out of my way to make my program more robust for my future self. Given the deadlines for this assignment I couldn't make all the necessary improvements I wanted, so I learned from this how to prioritize the most important components.

Issues

Despite the growth I felt in this assignment, there were a couple unresolved issues. The first was a segmentation fault that occurred whenever the window is resized. This happens since none of my calculations in the code account for a window resize. However, since this does not affect the game play and the window starts off in the correct size, I did not patch it. The other issue was that my scaling for the checkerboard was slightly off, so it was a little larger than the window. This issue was so small that it is almost unnoticeable, which is how it escaped my gaze.



Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = main.o Checkers.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system
DEPS = Checkers.hpp

all: Checkers_driver lint

Checkers_driver: $(OBJ)
	$(CC) $(CFLAGS) -o Checkers_driver $(OBJ) $(LIBS)
lint:
	cpplint *.cpp Headers/Checkers.hpp
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) Checkers_driver lint

```

main.cpp

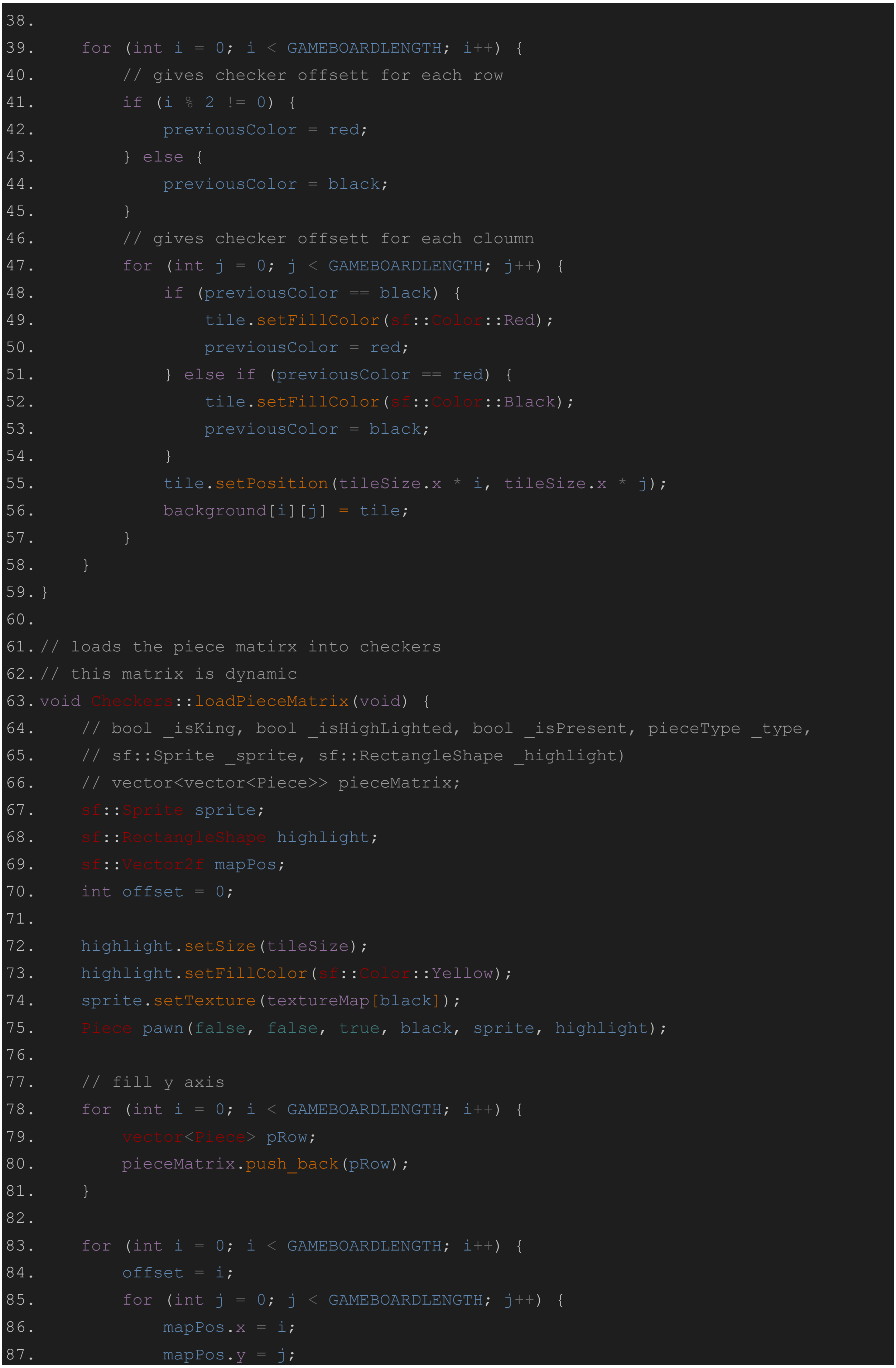
```
/* "Copyright [2023] <Daniel Bergeron>" */
#include "Headers/Checkers.hpp"

void clearBuffer(void);

int main(int argc, char* argv[]) {
    sf::RenderWindow window1(sf::VideoMode(GAMEBOARDLENGTH * SPACESIZES,
                GAMEBOARDLENGTH * SPACESIZES), "CHECKERS");
    Checkers game1;
    while (window1.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
        if (event.type == sf::Event::Closed) {
            clearBuffer();
            window1.close();
        }
    }
        if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
            // left mouse button is pressed:
            game1.highLight(sf::Mouse::getPosition(window1));
        }
        window1.clear(sf::Color::Black);
        window1.draw(game1);
        window1.display();
    }
    return 0;
}

// removes junk input from buffer so next run doesn't
// segment fault
void clearBuffer(void) {
    while (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {}
}


```

Checkers.hpp

```
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS4A_HEADERS_CHECKERS_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS4A_HEADERS_CHECKERS_HPP_
// Copyright [2023] <Daniel Bergeron>
// ptree class that derives from sf::Drawable
#include <iostream>
using std::cout;
using std::endl;
#include <utility>
using std::pair;
#include <string>
using std::string;
using std::stoi;
using std::stod;
#include <vector>
using std::vector;
#include <map>
using std::map;
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

// constants for conversions and intial set up
const int GAMEBOARDLENGTH = 8;
const int SPACESIZES = 64;
const int UPPERBOARDLIMIT = 2;
const int LOWERBOARDLIMIT = 5;

// type for pieces
enum pieceType {red, black, redKing, blackKing};

class Piece {
 public:
    // constructors
    Piece() : isKing(false), isHighLighted(false), type(red),
                                isPresent(false) {}
    Piece(bool _isKing, bool _isHighLighted, bool _isPresent, pieceType _type,
            sf::Sprite _sprite, sf::RectangleShape _highlight);
    // accessors
    inline bool getIsPresent(void) {return isPresent; }
    inline bool getIsKing(void) {return isKing; }
    inline bool getIsHighLighted(void) {return isHighLighted; }
    inline pieceType getpieceType(void) {return type; }
    inline sf::Sprite getSprite(void) {return sprite; }
    inline sf::RectangleShape getHighLighted(void) {return highlight; }
    inline sf::Vector2f getPosition(void) {return position; }

    // muttators
    inline void setIsKing(bool _isKing) {isKing = _isKing; }
    inline void setIsHighLighted(bool _isSelected)
                                 {isHighLighted = _isSelected; }
    inline void setType(pieceType _type) {type = _type; }
    inline void setSprite(sf::Sprite _sprite) {sprite = _sprite; }
    inline void setHighLight(sf::RectangleShape _selected)
                                 {highlight = _selected; }
    inline void setIsPresent(bool _isPresent) {isPresent = _isPresent; }
    inline void setPosition(sf::Vector2f _position) {position = _position; }

 private:
    // bool vars tell if specifc piece attributes are ture or not
    bool isKing;
    bool isHighLighted;
    bool isPresent;
    // other mem vars
    pieceType type;
    sf::Sprite sprite;
    sf::RectangleShape highlight;
    sf::Vector2f position;
};

class Checkers : public sf::Drawable {
 public:
    // constructor
    Checkers();
    // tells if checkers class to enable a pieces highlight attribute
    void highLight(sf::Vector2i localPosition);

    // accessors
 private:
    // helper & private funcs
    virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;

    // load intial elements
    void loadTextureMap(void);
    void loadPieceMatrix(void);
    void loadBackground(void);

    // member vars
    sf::Vector2f tileSize;
    sf::Vector2f currHighLight;
    sf::RectangleShape background[GAMEBOARDLENGTH][GAMEBOARDLENGTH];
    vector<vector<Piece>> pieceMatrix;
    map<pieceType, sf::Texture> textureMap;
};

#endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS4A_HEADERS_CHECKERS_HPP_


```

Checkers.cpp

```
// "Copyright [2023] Daniel Bergeron"
#include "Headers/Checkers.hpp"

// consturctors
Piece::Piece(bool _isKing, bool _isHighLighted, bool _isPresent,
        pieceType _type, sf::Sprite _sprite, sf::RectangleShape _highlight) {
    isKing = _isKing;
    isHighLighted = _isHighLighted;
    type = _type;
    sprite = _sprite;
    highlight = _highlight;
    isPresent = _isPresent;
}

Checkers::Checkers() {
    tileSize.x = static_cast<float>(SPACESIZES);
    tileSize.y = tileSize.x;

    currHighLight.x = -1;
    currHighLight.y = -1;

    // load background()
    loadBackground();

    // loadstextures into map
    loadTextureMap();

    // adds pieces to matrix
    loadPieceMatrix();
}

// helper functions for constructor
// loads static tile background
void Checkers::loadBackground(void) {
    sf::RectangleShape tile;
    pieceType previousColor = black;
    tile.setSize(tileSize);
```



```

    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        // gives checker offsett for each row
        if (i % 2 != 0) {
            previousColor = red;
        } else {
            previousColor = black;
        }
        // gives checker offsett for each cloumn
        for (int j = 0; j < GAMEBOARDLENGTH; j++) {
            if (previousColor == black) {
                tile.setFillColor(sf::Color::Red);
                previousColor = red;
            } else if (previousColor == red) {
                tile.setFillColor(sf::Color::Black);
                previousColor = black;
            }
            tile.setPosition(tileSize.x * i, tileSize.x * j);
            background[i][j] = tile;
        }
    }
}

// loads the piece matirx into checkers
// this matrix is dynamic
void Checkers::loadPieceMatrix(void) {
    // bool _isKing, bool _isHighLighted, bool _isPresent, pieceType _type,
    // sf::Sprite _sprite, sf::RectangleShape _highlight)
    // vector<vector<Piece>> pieceMatrix;
    sf::Sprite sprite;
    sf::RectangleShape highlight;
    sf::Vector2f mapPos;
    int offset = 0;

    highlight.setSize(tileSize);
    highlight.setFillColor(sf::Color::Yellow);
    sprite.setTexture(textureMap[black]);
    Piece pawn(false, false, true, black, sprite, highlight);

    // fill y axis
    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        vector<Piece> pRow;
        pieceMatrix.push_back(pRow);
    }

    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        offset = i;
        for (int j = 0; j < GAMEBOARDLENGTH; j++) {
            mapPos.x = i;
            mapPos.y = j;
        pawn.setPosition(mapPos);
        if (j > UPPERBOARDLIMIT) {
                pawn.setType(red);
                sprite.setTexture(textureMap[red]);
                pawn.setSprite(sprite);
        } else {
            pawn.setType(black);
            sprite.setTexture(textureMap[black]);
            pawn.setSprite(sprite);
        }
        if (j > UPPERBOARDLIMIT && j < LOWERBOARDLIMIT) {
            pawn.setIsPresent(false);
            } else {
                (offset % 2 == 0) ? pawn.setIsPresent(false)
                        :pawn.setIsPresent(true);
            }
            pieceMatrix[i].push_back(pawn);
            offset++;
        }
    }
}

// loads a texture map for sprites to
// acces since they having a lot of textures
// is heavy
void Checkers::loadTextureMap(void) {
    sf::Texture texture;
    texture.loadFromFile("checkers/blackking.png");
    textureMap[blackKing] = texture;
    texture.loadFromFile("checkers/blackpawn.png");
    textureMap[black] = texture;
    texture.loadFromFile("checkers/redking.png");
    textureMap[redKing] = texture;
    texture.loadFromFile("checkers/redpawn.png");
    textureMap[red] = texture;
}

// draw function is a overrides of sfml drawable
void Checkers::draw(sf::RenderTarget &target, sf::RenderStates states) const {
    Piece piece;
    sf::Vector2f posOffsett;
    sf::Sprite spritePiece;
    sf::RectangleShape highLight;
    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        for (int j = 0; j < GAMEBOARDLENGTH; j++) {
            target.draw(background[i][j], states);
            piece = pieceMatrix[i][j];
            if (piece.getIsPresent() == true) {
                posOffsett = piece.getPosition();
```

```
                    posOffsett.x = posOffsett.x * SPACESIZES;
                    posOffsett.y = posOffsett.y * SPACESIZES;
                    if (piece.getIsHighLighted() == true) {
                        highLight = piece.getHighLighted();
                        highLight.setPosition(posOffsett);
                        target.draw(highLight, states);
                    }
                    spritePiece = piece.getSprite();
                    spritePiece.setPosition(posOffsett);
                    target.draw(spritePiece, states);
                }
            }
        }
        return;
    }

    // is passed a moses position in pixel space
    // highlights the piece in tile space
    void Checkers::highLight(sf::Vector2i localPosition) {
        // highlight current selection
        Piece selected = pieceMatrix[localPosition.x / SPACESIZES]
                    [localPosition.y / SPACESIZES];
        if (selected.getIsPresent() == true) {
            selected.setIsHighLighted(true);
        }
        pieceMatrix[localPosition.x / SPACESIZES]
                    [localPosition.y / SPACESIZES] = selected;

        // remove previous selection
        if (currHighLight == selected.getPosition()) {
            return;
        }
        if (currHighLight.x >= 0 && currHighLight.y >= 0) {
            pieceMatrix[currHighLight.x][currHighLight.y].setIsHighLighted(false);
        }
        currHighLight.x = selected.getPosition().x;
        currHighLight.y = selected.getPosition().y;
        return;
    }

```

PS4B – Checkers

Program Discussion

The goal of this assignment was to build of the previous assignments' implementation of a checkers board and add the mechanics. I was able to bring the pieces to life, giving them movement and adding the rules that the movement had to adhere to. I also added a win condition for each side. This program took the interphase I developed in the first assignment and utilized it to great effect. Compared to the previous assignment, there were a lot of difficult decisions and issues I needed to overcome. At this point I came across another object-oriented programing design mistake. While every piece in the game had the power to move, the movement depended on the type, of which there were two. King pieces and pawn pieces. It was at this point that I should have modified my old code and created two new classes to interphase with. The first would be the pawn class and the second would be the king class. Both classes should have been inherited from the piece class. Which would have been abstract. With this I could have had each child's class manage the rules of movement for the pieces rather than the functions I created. If I took this route, it would have wider implications throughout my program, and would have required a complete rework of the entire program structure. Given that the deadline for these assignments is quite strict I decided to stick with what I had. This created more complicated functions, but in the end, it was able to accomplish its tasks. I created a playable checkers game.

Key Elements Discussion

For data structures I used maps & vectors. I used vectors because of their built-in interphase along with its ability to make a marix and model a 2-d surface in a program. Maps where use full for encoding and decoding, movement/texture information that I needed throughout the program, since the amount of data for these two cases was less important than the type of data, the maps' ability to use key allowed me to have vital info on hand. OOP in the checkers class I created a supported a piece class that could interphase with the checkers' class. This idea of interfacing was imperative for this project's success. For an algorithm & lambda I used a for_each range function which allowed me to apply a lambda function that calculated how many pieces were left on the board. Using the lambda here allowed me to remove a nested loop and make the program more readable. I addition to these tools I had to develop my own logical structure on how the code should run. So, I came up with some important logic.

In this program as stated before I have a checkers class, this class is responsible for visuals along with enforcing game rules on those visuals. It accomplishes this with three main functions an overridden draw function from sf::drawable, this function is solely responsible for drawing everything. No other function is allowed to make anything visible to the users without going through the draw function. There is a move function which breaks the problem of movement down into several important aspects, first can we move, if so what type of move can we do, then if possible execute the move, if move succeeded, then we no longer have a selected piece and finally now that we moved who gets to move next; (in hindsight the turns could have been separate function, a lot I would change with more time). Along with a couple accessor functions the next important function is the endgame check, this function which is responsible for checking if the game can move to an endgame state. The Checkers class then interphases with the piece class, which contains information on all the attributes of a piece object. As stated in the previous discussion it is here, I would have broken up the class. However, the piece class develops a strong enough interphase that allows checkers to use its methods to build a sturdy game.

What I Learned

I learned in this program the importance of adhering to the design principles of the paradigm. This code would have been much shorter and more robust if had adhered more to Object-Oriented design principles. However, this program was a significant step in the right direction compared to previous programs.

Issues

This program retains the same issues as the previous program, yet it did not cause any issues in this implementation. The main issue faced in this program had to do with the input from the mouse. Input buffering was difficult to manage, causing segmentation faults to occur as well as selection delays. The segmentation faults were caused by the same issue in the previous assignment. Yet it became slightly more relevant in this assignment because the players are selecting and moving pieces frequently which may accidentally cause them to touch the window edge slightly resizing it. This overtime with many moves creates a banned around the game board of bad positional data. If a player accidentally selects in this area, it causes the program to have a segmentation fault.

Code

Makefile

```makefile
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = main.o Checkers.o
LIBS = -lsfml-window -lsfml-graphics -lsfml-system
DEPS = Checkers.hpp

all: Checkers lint

Checkers: $(OBJ)
	$(CC) $(CFLAGS) -o Checkers $(OBJ) $(LIBS)
lint:
	cpplint *.cpp Headers/Checkers.hpp
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) Checkers lint

```

main.cpp

```
/* "Copyright [2023] <Daniel Bergeron>" */
#include "Headers/Checkers.hpp"

void clearBuffer(void);

int main(int argc, char* argv[]) {
    sf::RenderWindow window1(sf::VideoMode(GAMEBOARDLENGTH * SPACESIZES,
                GAMEBOARDLENGTH * SPACESIZES), "CHECKERS");
    Checkers game1;
    sf::Vector2i location;

    // fonts
    sf::Text text, text2;
    sf::Font font;
    if (!font.loadFromFile("resources/endGameFont.ttf")) {
        cout << "Font failed to load\n";
    }
    text.setFont(font);
    text.setString("Game Over!!");
    text.setCharacterSize(100);
    text.setPosition((static_cast<float>(window1.getSize().x / 5.00)),
                (static_cast<float>(window1.getSize().y / 4.00)));
    text.setFillColor(sf::Color::White);
    text.setStyle(sf::Text::Bold | sf::Text::Underlined);

    text2.setFont(font);
    text2.setString("Press Q to quit");
    text2.setCharacterSize(30);
    text2.setFillColor(sf::Color::Blue);
    text2.setStyle(sf::Text::Bold | sf::Text::Underlined);

    clearBuffer();
    while (window1.isOpen()) {
        sf::Event event;
        while (window1.pollEvent(event)) {
            if (event.type == sf::Event::Closed) {
                clearBuffer();
                window1.close();
            }
        }
        // tells if its a move or not based on if present
        if (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {
            // left mouse button is pressed:
            game1.highLight(sf::Mouse::getPosition(window1));
            location = sf::Mouse::getPosition(window1);
            location.x /= SPACESIZES;
            location.y /= SPACESIZES;
            game1.move(location);
        }
        while (game1.endGameCheck()) {
            // inside the main loop, between window.clear()
            // and window.display()
            while (window1.pollEvent(event)) {
                if (event.type == sf::Event::Closed) {
                    clearBuffer();
                    window1.close();
                    return 0;
                }
            }
            if (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {
                return 0;
            }
            window1.clear(sf::Color::Black);
            window1.draw(game1);
            if (game1.getBlackPieceCount() == 0) {
                text.setString("RED WINS!!!");
            } else if (game1.getRedPieceCount() == 0) {
                text.setString("Black WINS!!!");
            } else {
                // For possible edge case
                text.setString("Draw!");
            }
            window1.draw(text);
            window1.draw(text2);
            window1.display();
        }
        // take selected input and see if next position clicked position is good
        // then move

        window1.clear(sf::Color::Black);
        window1.draw(game1);
        window1.display();
    }
    // clearBuffer();
    return 0;
}

// removes junk input from buffer so next run doesn't
// segment fault
void clearBuffer(void) {
    while (sf::Mouse::isButtonPressed(sf::Mouse::Left)) {}
    while (sf::Keyboard::isKeyPressed(sf::Keyboard::Q)) {}
```

```
}
```

Checkers.hpp

```
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS4B_HEADERS_CHECKERS_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS4B_HEADERS_CHECKERS_HPP_
// Copyright [2023] <Daniel Bergeron>
// ptree class that derives from sf::Drawable
#include <ranges>
#include <cmath>
using std::abs;
#include <iostream>
using std::cout;
using std::endl;
#include <utility>
using std::pair;
#include <algorithm>
#include <string>
using std::string;
using std::stoi;
using std::stod;
#include <vector>
using std::vector;
#include <map>
using std::map;
#include <SFML/System.hpp>
#include <SFML/Window.hpp>
#include <SFML/Graphics.hpp>

// constants for conversions and intial set up
const int GAMEBOARDLENGTH = 8;
const int SPACESIZES = 64;
const int UPPERBOARDLIMIT = 2;
const int LOWERBOARDLIMIT = 5;

// type for pieces
enum pieceType {empty, red, redKing, black, blackKing};
enum moveSet { nil, upLeft = 1, upLeftJump = -1, upRight = 2,
upRightJump = -2, downRight = 3, downRightJump = -3,
downLeft = 4, downLeftJump = -4};

class Piece {
 public:
    // constructors
    Piece() : isKing(false), isHighLighted(false), type(red),
                                 isPresent(false) {}
    Piece(bool _isKing, bool _isHighLighted, bool _isPresent, pieceType _type,
            sf::Sprite _sprite, sf::RectangleShape _highlight);

    // accessors
    inline bool getIsPresent(void) {return isPresent; }
    inline bool getIsKing(void) {return isKing; }
    inline bool getIsHighLighted(void) {return isHighLighted; }
    inline pieceType getpieceType(void) {return type; }
    inline sf::Sprite getSprite(void) {return sprite; }
    inline sf::RectangleShape getHighLighted(void) {return highlight; }
    inline sf::Vector2f getPosition(void) {return position; }

    // muttators
    inline void setIsKing(bool _isKing) {isKing = _isKing; }
    inline void setIsHighLighted(bool _isSelected)
                                 {isHighLighted = _isSelected; }
    inline void setType(pieceType _type) {type = _type; }
    inline void setSprite(sf::Sprite _sprite) {sprite = _sprite; }
    inline void setHighLight(sf::RectangleShape _selected)
                                 {highlight = _selected; }
    inline void setIsPresent(bool _isPresent) {isPresent = _isPresent; }
    inline void setPosition(sf::Vector2f _position) {position = _position; }

 private:
    // bool vars tell if specifc piece attributes are ture or not
    bool isKing;
    bool isHighLighted;
    bool isPresent;
    // other mem vars
    pieceType type;
    sf::Sprite sprite;
    sf::RectangleShape highlight;
    sf::Vector2f position;
};

class Checkers : public sf::Drawable {
 public:
    // constructor
    Checkers();
    // tells if checkers class to enable a pieces highlight attribute
    void highLight(sf::Vector2i localPosition);

    // move function
    bool move(sf::Vector2i LocalPosition);

    // turn management
    inline void increaseTurn(void) {turn++; }
    int getTurn(void);

    // Tells weather or not to end the game
    bool endGameCheck();

    // accesors
    inline int getRedPieceCount(void) {return redWin; }
    inline int getBlackPieceCount(void) {return blackWin; }
```

```

private:
        // helper & private funcs
        virtual void draw(sf::RenderTarget& target, sf::RenderStates states) const;

        // move helper functions
        moveSet getMovetype(sf::Vector2i LocalPosition);
        // function resets the highlighted square
        void resetHighLight(void);

        // when given a move it is responsible
        bool executeMove(moveSet currMove);

        // execute helper
        bool removePiece(moveSet destroy);

        // function converts a normal pawn to king
        // Needs the location and the type to do it
        void makeKing(pieceType currType, sf::Vector2i LocalPosition);

        // load intial elements
        void loadTextureMap(void);
        void loadPieceMatrix(void);
        void loadBackground(void);
        void loadMoveMap(void);

        // member vars
        int turn = 0;
        int redWin = 0;
        int blackWin = 0;
        Piece currPiece;
        sf::Vector2f tileSize;
        sf::Vector2f currHighLight;
        sf::RectangleShape background[GAMEBOARDLENGTH][GAMEBOARDLENGTH];
        vector<vector<Piece>> pieceMatrix;
        map<pieceType, sf::Texture> textureMap;
        map<moveSet, sf::Vector2i> moveMap;
    };

    #endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS4B_HEADERS_CHECKERS_HPP_


```

Checkers.cpp

```
// "Copyright [2023] Daniel Bergeron"
#include "Headers/Checkers.hpp"

// consturctors
Piece::Piece(bool _isKing, bool _isHighLighted, bool _isPresent,
        pieceType _type, sf::Sprite _sprite, sf::RectangleShape _highlight) {
    isKing = _isKing;
    isHighLighted = _isHighLighted;
    type = _type;
    sprite = _sprite;
    highlight = _highlight;
    isPresent = _isPresent;
}

Checkers::Checkers() {
    tileSize.x = static_cast<float>(SPACESIZES);
    tileSize.y = tileSize.x;

    currHighLight.x = -1;
    currHighLight.y = -1;

    // load background()
    loadBackground();

    // loadstextures into map
    loadTextureMap();

    // adds pieces to matrix
    loadPieceMatrix();

    // loads moves into map
    loadMoveMap();
}

void Checkers::loadMoveMap(void) {
    // enum moveSet {nil, upLeft, upRight, downLeft, downRight, upLeftJump,
    // upRightJump, downRightJump, downLeftJump };
    sf::Vector2i moveVal;
    moveVal.x = 0; moveVal.y = 0;
    moveMap[nil] = moveVal;
    moveVal.x = -1; moveVal.y = -1;
    moveMap[upLeft] = moveVal;
    moveVal.x = 1; moveVal.y = -1;
    moveMap[upRight] = moveVal;
    moveVal.x = 1; moveVal.y = 1;
    moveMap[downLeft] = moveVal;
    moveVal.x = -1; moveVal.y = 1;
    moveMap[downRight] = moveVal;
    moveVal.x = -2; moveVal.y = -2;
    moveMap[upLeftJump] = moveVal;
    moveVal.x = 2; moveVal.y = -2;
    moveMap[upRightJump] = moveVal;
    moveVal.x = -2; moveVal.y = 2;
    moveMap[downRightJump] = moveVal;
    moveVal.x = 2; moveVal.y = 2;
    moveMap[downLeftJump] = moveVal;
    return;
}

// helper functions for constructor
// loads static tile background
void Checkers::loadBackground(void) {
    sf::RectangleShape tile;
    pieceType previousColor = black;
    tile.setSize(tileSize);

    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        // gives checker offsett for each row
        if (i % 2 != 0) {
            previousColor = red;
        } else {
            previousColor = black;
        }
        // gives checker offsett for each cloumn
        for (int j = 0; j < GAMEBOARDLENGTH; j++) {
            if (previousColor == black) {
                tile.setFillColor(sf::Color::Red);
                previousColor = red;
            } else if (previousColor == red) {
                tile.setFillColor(sf::Color::Black);
                previousColor = black;
            }
            tile.setPosition(tileSize.x * i, tileSize.x * j);
            background[i][j] = tile;
        }
    }
}

// loads the piece matirx into checkers
// this matrix is dynamic
void Checkers::loadPieceMatrix(void) {
    // bool _isKing, bool _isHighLighted, bool _isPresent, pieceType _type,
    // sf::Sprite _sprite, sf::RectangleShape _highlight)
    // vector<vector<Piece>> pieceMatrix;
    sf::Sprite sprite;
    sf::RectangleShape highlight;
    sf::Vector2f mapPos;
    int offset = 0;

        highlight.setSize(tileSize);
        highlight.setFillColor(sf::Color::Yellow);
        sprite.setTexture(textureMap[black]);
        Piece pawn(false, false, true, black, sprite, highlight);

        // fill y axis
```

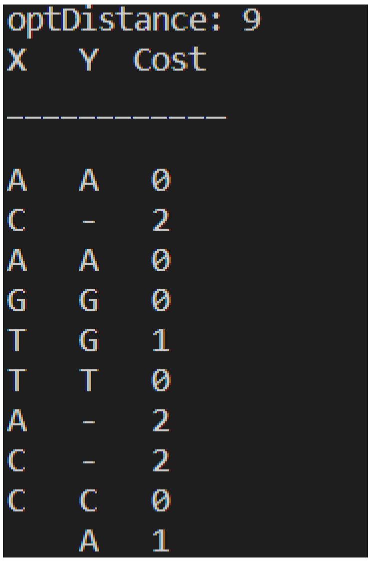
```cpp
        for (int i = 0; i < GAMEBOARDLENGTH; i++) {
            vector<Piece> pRow;
            pieceMatrix.push_back(pRow);
        }

        for (int i = 0; i < GAMEBOARDLENGTH; i++) {
            offset = i;
            for (int j = 0; j < GAMEBOARDLENGTH; j++) {
                mapPos.x = i;
                mapPos.y = j;
                pawn.setPosition(mapPos);
                if (j > UPPERBOARDLIMIT) {
                        pawn.setType(red);
                        sprite.setTexture(textureMap[red]);
                        pawn.setSprite(sprite);
                } else {
                    pawn.setType(black);
                    sprite.setTexture(textureMap[black]);
                    pawn.setSprite(sprite);
                }
                if (j > UPPERBOARDLIMIT && j < LOWERBOARDLIMIT) {
                    pawn.setType(empty);
                    pawn.setIsPresent(false);
                } else {
                    (offset % 2 == 0) ? pawn.setIsPresent(false)
                            :pawn.setIsPresent(true);
                }
                pieceMatrix[i].push_back(pawn);
                offset++;
            }
        }
    }

    // loads a texture map for sprites to
    // acces since they having a lot of textures
    // is heavy
    void Checkers::loadTextureMap(void) {
        sf::Texture texture;
        texture.loadFromFile("resources/blackking.png");
        textureMap[blackKing] = texture;
        texture.loadFromFile("resources/blackpawn.png");
        textureMap[black] = texture;
        texture.loadFromFile("resources/redking.png");
        textureMap[redKing] = texture;
        texture.loadFromFile("resources/redpawn.png");
        textureMap[red] = texture;
    }

    // draw function is a overrides of sfml drawable
```

```cpp
void Checkers::draw(sf::RenderTarget &target, sf::RenderStates states) const {
    Piece piece;
    sf::Vector2f posOffsett;
    sf::Sprite spritePiece;
    sf::RectangleShape highLight;
    for (int i = 0; i < GAMEBOARDLENGTH; i++) {
        for (int j = 0; j < GAMEBOARDLENGTH; j++) {
            target.draw(background[i][j], states);
            piece = pieceMatrix[i][j];
            if (piece.getIsPresent() == true) {
                posOffsett = piece.getPosition();
                posOffsett.x = posOffsett.x * SPACESIZES;
                posOffsett.y = posOffsett.y * SPACESIZES;
                if (piece.getIsHighLighted() == true) {
                    highLight = piece.getHighLighted();
                    highLight.setPosition(posOffsett);
                    target.draw(highLight, states);
                }
                spritePiece = piece.getSprite();
                spritePiece.setPosition(posOffsett);
                target.draw(spritePiece, states);
            }
        }
    }
    return;
}

// is passed a moses position in pixel space
// highlights the piece in tile space
void Checkers::highLight(sf::Vector2i localPosition) {
    // highlight current selection
    int colorTurn = getTurn();
    Piece selected = pieceMatrix[localPosition.x / SPACESIZES]
                [localPosition.y / SPACESIZES];
    // if piece exsits highlight else do nothing
    if (selected.getIsPresent() == true) {
        if (colorTurn == 0 && selected.getpieceType() >= black) {
            return;
        } else if (colorTurn == 1 && selected.getpieceType() < black) {
            return;
        }
    } else {
        return;
    }
    selected.setIsHighLighted(true);

    pieceMatrix[localPosition.x / SPACESIZES]
                [localPosition.y / SPACESIZES] = selected;

    // remove previous selection
    if (currHighLight == selected.getPosition()) {
        currPiece = selected;
        return;
    }
    if (currHighLight.x >= 0 && currHighLight.y >= 0) {
        pieceMatrix[currHighLight.x][currHighLight.y].setIsHighLighted(false);
    }
    currHighLight.x = selected.getPosition().x;
    currHighLight.y = selected.getPosition().y;
    currPiece = selected;
    return;
}

// is already converted to tile space
bool Checkers::move(sf::Vector2i LocalPosition) {
    // first check if piece at location is present
    // or if it is the selected piece
    Piece newPiece = pieceMatrix[LocalPosition.x][LocalPosition.y];
    if (newPiece.getIsPresent() == true
                || newPiece.getIsHighLighted() == true) {
        return false;
    }

    // call function to seee if local position matches anything in moveset
    // if so return move to use
    moveSet currMove;
    currMove = getMovetype(LocalPosition);
    if (currMove == nil) {
        return false;
    }
    // check if king
    // if king all moves from here are valid
    // call a function to execute move if
    // if king condition fails move based on color
    // get piece color & branch based off piece color
    // red only moves up and black down
    // inside color if's
    pieceType currType = currPiece.getpieceType();
    if (currType == red) {
        if (currMove == downLeft || currMove == downRight
            || currMove == downLeftJump || currMove == downRightJump) {
                return false;
        }
    } else if (currType == black) {
        if (currMove == upLeft || currMove == upRight
            || currMove == upLeftJump || currMove == upRightJump) {
                return false;
        }
    }
    resetHighLight();
    // if execute happens piece is move into Local position on matrix
    // if executed then new turn
    if (executeMove(currMove)) {
        increaseTurn();
    }

    // after move, check if piece is in the bottom(black)
    // or top(red) if so call king function
    if ((currType == black && LocalPosition.y == 7)
            || (currType == red && LocalPosition.y == 0)) {
        makeKing(currType, LocalPosition);
    }

    // return true if move worked
    return true;
}

// goal of function is to validate the seleceted space
// need to figure out where curr piece is in regards to newSpace
moveSet Checkers::getMovetype(sf::Vector2i LPosition) {
    Piece emptySpace;
    moveSet move = nil;
    sf::Vector2i offset;
    offset.x = currPiece.getPosition().x;
    offset.y = currPiece.getPosition().y;

    // 1) use map to store what each move does mathamtically
    // 2) using the curr pieces positon use the pair retieved from
    // map add x & y together if the the sums agree return move type

    map<moveSet, sf::Vector2i>::iterator m;
    for (m = moveMap.begin(); m != moveMap.end(); ++m) {
        offset.x += m->second.x;
        offset.y += m->second.y;
        move = m->first;
        if (offset == LPosition) {
            return move;
        } else {
            offset.x = currPiece.getPosition().x;
            offset.y = currPiece.getPosition().y;
            move = nil;
        }
    }
    return move;
}

// resest highlight for next draw cycle
void Checkers::resetHighLight(void) {
    Piece selected = pieceMatrix[currHighLight.x][currHighLight.y];
    selected.setIsHighLighted(false);
    pieceMatrix[currHighLight.x][currHighLight.y] = selected;
    return;
}

bool Checkers::executeMove(moveSet currMove) {
    // if move is jump remove middle piece
    // jumps are negative since piece will be taken
    // so go to a normal move space in that direction
    // and check if piece is present if not move fails
    bool failedJumped;
    if (currMove < 0) {
        failedJumped = removePiece(static_cast<moveSet>(abs(currMove)));
        if (failedJumped) {
            cout << "jump failed\n";
            return false;
        }
    }
    Piece newPiece;
    // set new position
    sf::Vector2i pos = moveMap[currMove];
    pos.x += currPiece.getPosition().x;
    pos.y += currPiece.getPosition().y;

    newPiece = currPiece;
    newPiece.setPosition(static_cast<sf::Vector2f>(pos));
    newPiece.setIsHighLighted(false);
    // enable new piece with same attributes as currpiece
    pieceMatrix[pos.x][pos.y] = newPiece;
    // disable currPiece
    currPiece.setIsPresent(false);
    currPiece.setType(empty);
    currPiece.setIsHighLighted(false);
    pieceMatrix[currPiece.getPosition().x][currPiece.getPosition().y]
        = currPiece;
    return true;
}

void Checkers::makeKing(pieceType currType, sf::Vector2i pos) {
    sf::Sprite sprite;
    Piece king =  pieceMatrix[pos.x][pos.y];
    int x = currType;
    x++;
    currType = static_cast<pieceType>(x);
    king.setType(currType);
    king.setIsKing(true);
    sprite.setTexture(textureMap[currType]);
    king.setSprite(sprite);

    pieceMatrix[pos.x][pos.y] = king;
}

bool Checkers::removePiece(moveSet destroyMove) {
    // need moving pieces to test come back to later

    sf::Vector2i pos = moveMap[destroyMove];
    pos.x += currPiece.getPosition().x;
    pos.y += currPiece.getPosition().y;
    if ( pieceMatrix[pos.x][pos.y].getIsPresent() == false ) {
        return true;
    }
    // checks to see if they are jumping their own pawn
    pieceType currType = currPiece.getpieceType();
    if (currPiece.getIsKing()) {
        if ( pieceMatrix[pos.x][pos.y].getpieceType() == currType
        || pieceMatrix[pos.x][pos.y].getpieceType() == currType - 1) {
            return true;
        }
    } else {
        if ( pieceMatrix[pos.x][pos.y].getpieceType() == currType
            || pieceMatrix[pos.x][pos.y].getpieceType() == currType + 1) {
            return true;
        }
    }
    pieceMatrix[pos.x][pos.y].setIsPresent(false);
    pieceMatrix[pos.x][pos.y].setType(empty);
    return false;
}

int Checkers::getTurn(void) {
    return turn % 2;
}

// returns last color standing
bool Checkers::endGameCheck(void) {
    redWin = 0;
    blackWin = 0;
    vector<vector<Piece>>::iterator pieceVec = pieceMatrix.begin();
    vector<Piece>::iterator p;

    auto presentCheck = [&](Piece guy){
        if (guy.getIsPresent() == true) {
                if (guy.getpieceType() == red
                            || guy.getpieceType() == redKing) {
                    redWin++;
                } else if (guy.getpieceType() == black
                            || guy.getpieceType() == blackKing) {
                    blackWin++;
                }
            }
    };
    for ( ; pieceVec != pieceMatrix.end(); pieceVec++) {
        std::for_each(pieceVec->begin(), pieceVec->end(), presentCheck);
    }
    if (redWin == 0 || blackWin == 0) {
        return true;
    } else {
        return false;
    }
}


```

PS5 – DNA Sequence Alignment

Program Discussion

In this program, we learned about a programing paradigm called, dynamic programming. The goal was computing the optimal sequence alignment of two DNA strings. This program allows the user to figure out how closely related two DNA sequences are. I was able to accomplish this by building something called a cost matrix. Which uses a cost system to determine the optimal way through the matrix. To align the DNA, you can insert a gap which will add 2 to the cost, you can mismatch the DNA which will incur a cost of 1 or you have a match that incurs a cost of zero. These costs are laid out on a matrix, starting from the ends of the string to the top, or the bottom right corner of the matrix to the top left. Once the matrix is filled it can be traversed from the top left to the bottom right to find the optimal alignment. This was a difficult problem to solve and like other problems in this course it didn't require a lot of code, but it required a lot of thinking. To accomplish this task, I had to really break down the problem. To make sure I didn't get confused I heavily commented on the areas I didn't understand. Most of these comments took place in the header file so I could check back in on what the code for the function was implementing. The program faced many bugs when it came to the traversing of the matrix and the rules that governed that process. The main bug in this code that took me hours to decode didn't even come from the code but the files I inputted. Some of the files I input were not in Linux format. Instead, they were in windows format, so they had a '\r' invisible character in the input. So, when I used the get line function the character was being sucked into my string. This caused the ends of my DNA sequences to be warped and off since my program was considering an incorrect character. I was able to fix this by adding a guard in main that checked for the character and popped it out of the string if it saw it. I did this rather than changing the file type so the program could take both Linux and windows files. Once this issue was solved the program performed as intended.



```
optDistance: 9
X   Y  Cost
------------

A   A   0
C   -   2
A   A   0
G   G   0
T   G   1
T   T   0
A   -   2
C   -   2
C   C   0
    A   1
```

Terminal Output 1

Key Elements Discussion

The most important element of this program is the EDistance class and how its' methods interact with the cost matrix. The first of these key Elements is the constructor. The constructor is what builds the N x M array and initializes the values. The constructor also puts the true values in the farthest row and columns. This is because no matter the situation we know what those values are going to be. The next important method is `optDistance`. This is the Method uses dynamic programming to populate the matrix with cost values. The next important thing to do is to traverse the matrix based on these values to get the optimal distance value. The `alignment` method is what accomplishes this. The method goes through the matrix from the top left down choosing values based of three cases. Based of these cases a main string will be concatenated with a corresponding substring derived from the case itself. It’s important to note that if everything in this function is working properly it will always end in the bottom left corner or the zero condition. The complexity of this program comes from the nature of the problem, but the Object-Oriented Principles, as well as the dynamic programing I implemented here are what made this program manageable.

What I Learned

I learned two keys’ things from this program. The first was how to break down a problem can be made much easier in programing when you use a programing paradigm tailored for the problem. The second thing I learned was that when programing it is important to always be aware of the environment you are in, as well as what environments your program pulls from.

Code

Makefile

```
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = EDistance.o main.o
LIBS = -lsfml-system
DEPS = Header/EDistance.hpp

all: EDistance lint

EDistance: $(OBJ)
    $(CC) $(CFLAGS) -o EDistance $(OBJ) $(LIBS)
lint:
    cpplint *.cpp Header/EDistance.hpp
%.o: %.cpp $(DEPS)
    $(CC) $(CFLAGS) -c $<
clean:
    rm -f $(OBJ) EDistance lint

```

main.cpp

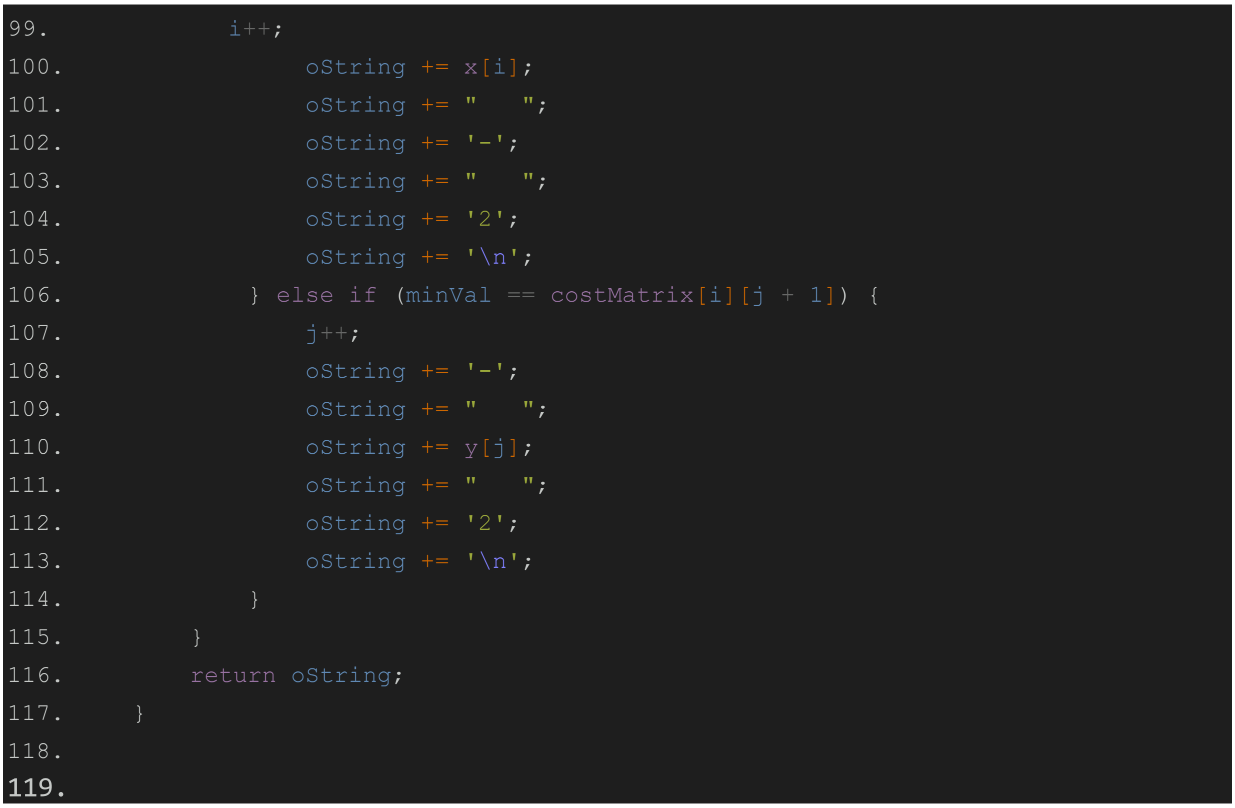
```
// Copyright [2023] <Daniel Bergeron>
#include "Header/EDistance.hpp"

int main(int argc, char *argv[]) {
    // input handling
    std::ifstream fp;
    fp.open(argv[1]);
    string input1, input2;
    getline(fp, input1);
    if (input1.back() == '\r') {
        input1.pop_back();
    }
    getline(fp, input2);
    // deals with windows carriage return
    if (input2.back() == '\r') {
        input2.pop_back();
    }
    fp.close();

    // Sequencing
    int x = 0;
    EDistance test(input1, input2);
    x = test.optDistance();
    cout << "optDistance: "<< x << endl;
    cout << "X   Y  Cost\n";
    cout << "____________\n\n";
    cout << test.alignment();
    return 0;
}


```

EDistance.hpp

```
// Copyright [2023] <Daniel Bergeron>
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS5_HEADER_EDISTANCE_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS5_HEADER_EDISTANCE_HPP_
#include <fstream>
#include <iostream>
using std::cout;
using std::cin;
using std::endl;
#include <string>
using std::string;
#include <vector>
using std::vector;
#include <algorithm>

class EDistance {
 public:
    EDistance(const string& ip1, const string& ip2);
    static int penalty(char a, char b);
    static int min(int a, int b, int c);
    int optDistance(void);
    string alignment(void);
 private:
    void testPrint() const;
    vector<vector<int>> costMatrix;
    string x;
    string y;
};

#endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS5_HEADER_EDISTANCE_HPP_


```

EDistance.cpp

```cpp
// Copyright [2023] <Daniel Bergeron>
#include "Header/EDistance.hpp"

void EDistance::testPrint() const {
    std::cout << "===" << std::endl;
    for (int i = 0; i < static_cast<int>(x.size() + 1); i++) {
        for (int j = 0; j < static_cast<int>(y.size() + 1); j++) {
            std::cout << " " << costMatrix[i][j] << " ";
        }
        std::cout << std::endl;
    }
    std::cout << "===" << std::endl;
}

EDistance::EDistance(const string& ip1, const string& ip2) {
    x = ip1;  // our x side
    y = ip2;  // our y side
    int size1 = ip1.length();
    int size2 = ip2.length();

    costMatrix.resize(size1 + 1);
    for (int i = 0; i < size1 + 1; i++) {
        costMatrix[i].resize(size2 + 1);
    }

    costMatrix[size1][size2] = 0;
    for (int i = size2 - 1; i >= 0; i--) {
        costMatrix[size1][i] = costMatrix[size1][i + 1] + 2;
    }

    for (int i = size1 - 1; i >= 0; i--) {
        costMatrix[i][size2] = costMatrix[i + 1][size2] + 2;
    }

    testPrint();
}

int EDistance::penalty(char a, char b) {
    if (a == b) {
        return 0;
    }
    return 1;
}
int EDistance::min(int a, int b, int c) {
    return std::min(std::min(a, b), c);
}
int EDistance::optDistance(void) {
     for (int i = costMatrix.size() - 2; i >= 0; i--) {
        for (int j = costMatrix[i].size() - 2; j >= 0; j--) {
            costMatrix[i][j] = min(costMatrix[i+1][j+1] + penalty(x[i], y[j]),
                                costMatrix[i+1][j] + 2, costMatrix[i][j+1] + 2);
        }
    }

     testPrint();

    return costMatrix[0][0];
}
string EDistance::alignment(void) {
    auto toChar = [](int cost) {
        return static_cast<char>(cost + '0');
    };
    string oString;  // output string

    // Resoureces
    // need a return string
    // cost vector
    // function to convert num to chars
    // main case
    int i = 0, j = 0;
    int minVal;
    int cost = 0;
    while (i <= static_cast<int>(x.size()-1)
                && j <= static_cast<int>(y.size()-1)) {
        // base case at costmatrix[0][0]
        // load directly into string, inset x[0], y[0], then num
        // most optimal alligment
        // letters are matched
        // therefore we put both letters in ouput string
        // insert char 0
        // push into cost matrix
        minVal = min(costMatrix[i+1][j+1],
                        costMatrix[i+1][j], costMatrix[i][j+1]);
        if (minVal == costMatrix[i+1][j+1]) {
            i++;
            j++;
            cost = penalty(x[i], y[j]);
            if(cost == 0 && (i > static_cast<int>(x.size()-1)
                        || j > static_cast<int>(x.size()-1))) {
                return oString;
            }
            oString += x[i];
            oString += "   ";
            oString += y[j];
            oString += "   ";
            oString += toChar(cost);
            oString += '\n';
        } else if (minVal == costMatrix[i+1][j]) {
```

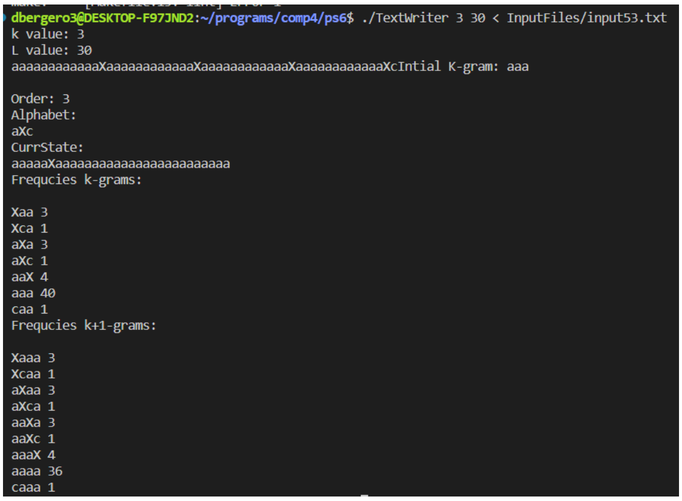


```
            i++;
                oString += x[i];
                oString += "    ";
                oString += '-';
                oString += "    ";
                oString += '2';
                oString += '\n';
            } else if (minVal == costMatrix[i][j + 1]) {
                j++;
                oString += '-';
                oString += "    ";
                oString += y[j];
                oString += "    ";
                oString += '2';
                oString += '\n';
            }
        }
        return oString;
    }


```

PS6 – Random Writer

Program Discussion

This program was a very conceptually difficult program to deal with as well as frustrating. This program took in large input file of text along with two numbers. The two numbers K is used to indicate the size of substrings, and L used to indicate the amount of random text to be generated. Then the file was read in. Once the information was read in the program constructed a randWriter using that information. The randWriter uses K to create every possible substring of size K with matching order. So, no substrings were constructed from random points in the input it was all sequential. These substrings referred to as Kgrams were then put in a map which were mapped to another map containing all the existing K+1 instances of themselves in the input string. The K+1 instances were then mapped to their number of occurrences. A method called `kRand` was called to create a random character using the Kgram and return it. This value is given to the `generate` which calls the KRand method L times. This will generate pseudo random character. It is here an important aspect of the program must be introduced. The occurrences of each character must match that of the original string. For example, if the input contains no 'a's then the generator can't make an 'a'. If there are 50 'a's to 20 'b's in the original text then the generate function has to match that ratio. I was able to accomplish this by taking the occurrences values of the k+1 grams and putting them on a theoretical number line where each k+1 grams domain matched the probability of the character appearing. The reason I used the K+1 grams is because they are the only thing in the program containing any information of what characters could come next. The three components of this program for me were trying to understand the problem, since the directions at time could be confusing. Mapping the Kgrams to their K+1 self and then mapping that to number of appearances. Finally creating the probability logic was a logical heavy challenge. I was able to test these functions extensively using my test driver. This ensured that they worked and that I accomplished my task.



```
dbergero3@DESKTOP-F97JND2:~/programs/comp4/ps6$ ./TextWriter 3 30 < InputFiles/input53.txt
k value: 3
L value: 30
aaaaaaaaaaaaXaaaaaaaaaaaaXaaaaaaaaaaaaXaaaaaaaaaaaaXcIntial K-gram: aaa

Order: 3
Alphabet:
aXc
CurrState:
aaaaaXaaaaaaaaaaaaaaaaaaaaaaaa
Frequcies k-grams:

Xaa 3
Xca 1
aXa 3
aXc 1
aaX 4
aaa 40
caa 1
Frequcies k+1-grams:

Xaaa 3
Xcaa 1
aXaa 3
aXca 1
aaXa 3
aaXc 1
aaaX 4
aaaa 36
caaa 1
```

Terminal Output 2

Key Elements Discussion

The three key elements of this program were the class system, its driver implementation, and the testing. The randWriter is encapsulated in its own class, this is important because it isolates two of the most difficult parts of this assignment. The rand method and the generator method. My main function needs to only call these methods and not have to worry about the complexity of the implementation. The next was the drivers I created for the project, one is the actual routine for randWriter and the other is for testing purposes. Due to the Object-Oriented nature of the program the first driver was very easy to build since it mostly just handled input and output(IO). The second driver was an isolated environment to try and break my methods. In this driver, I used the boost library to test my class object. Using boost is key because it helps you Isolate the issue in your program, by giving you good feedback.

What I Learned

This programing assignment helped to understand testing and why it's important. The program helped me once again in learning how to logically break down a complex problem into smaller more manageable bits.

Code

Makefile

```makefile
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = RandWriter.o test.o
OBJ2 = TextWriter.o RandWriter.o
LIBS = -lboost_unit_test_framework
DEPS = Header/RandWriter.hpp

all: TextWriter Test lint

TextWriter: $(OBJ2)
	$(CC) $(CFLAGS) -o TextWriter $(OBJ2) $(LIBS)
Test: $(OBJ)
	$(CC) $(CFLAGS) -o Test $(OBJ) $(LIBS)
lint:
	cpplint *.cpp Header/RandWriter.hpp
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) $(OBJ2) TextWriter Test lint

```

textWriter.cpp

```cpp
// Copyright [2023] <Daniel Bergeron>
#include "Header/RandWriter.hpp"

int main(int argc, char* argv[]) {
    string Kstr = argv[1];
    string Lstr = argv[2];
    int K = stoi(Kstr);
    int L = stoi(Lstr);
    std::cout << "k value: " << K << std::endl;
    std::cout << "L value: " << L << std::endl;

    string input, temp;
    while (!std::cin.eof()) {
        std::cin >> temp;
        input += temp;
    }
    std::cout << input;
    string seed;
    for (int i = 0; i < K; i++) {
        seed += input[i];
    }
    std::cout << "Intial K-gram: " << seed << std::endl;
    RandWriter writer(input, K);
    try {
       writer.generate(seed, L);
    } catch (const std::runtime_error& error) {
       std::cout << "Exception caught: " << error.what() << std::endl;
    }

    writer.operator<<(std::cout);
    return 0;
}

```

RandWriter.hpp

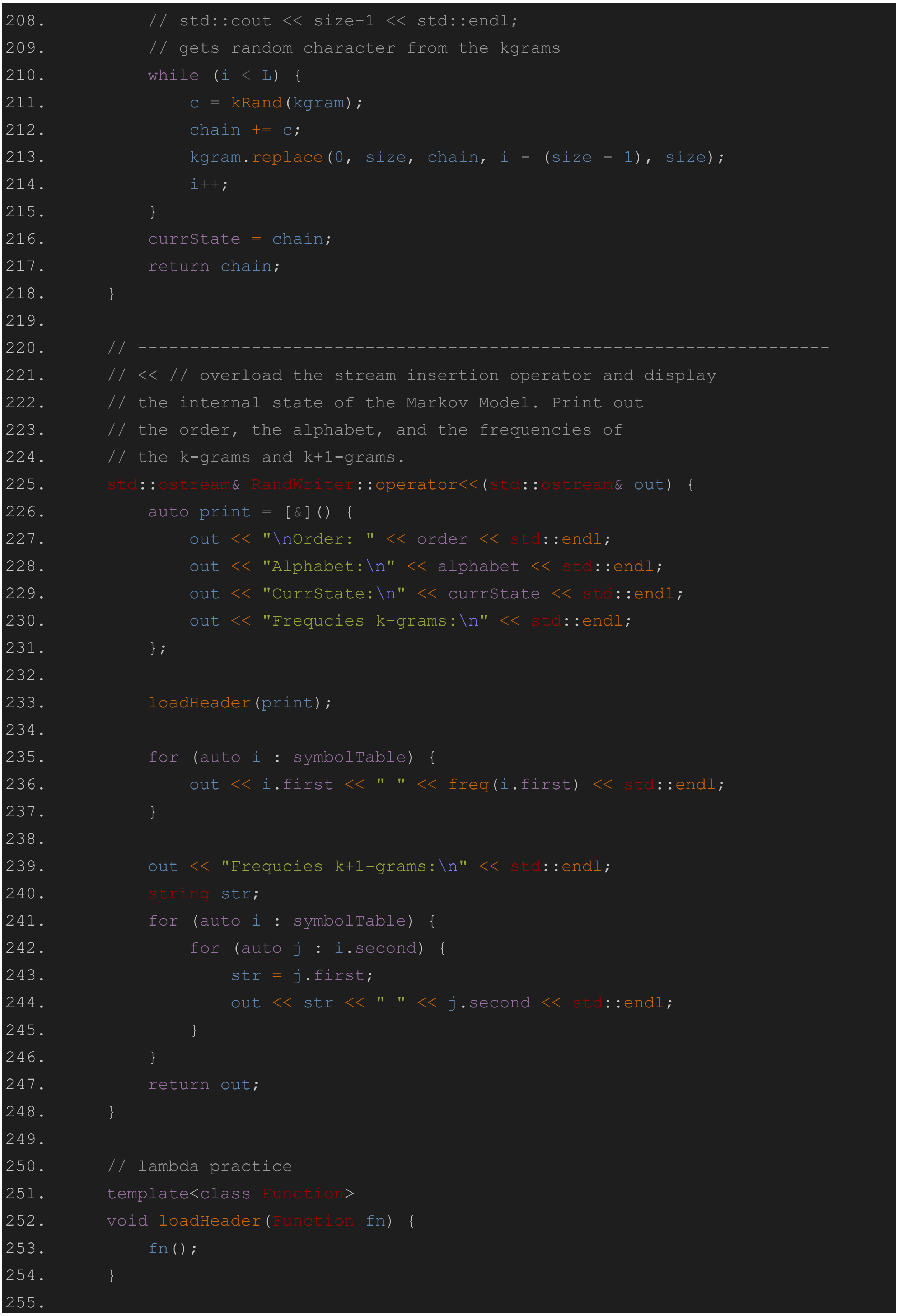
```
// Copyright [2023] <Daniel Bergeron>
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS6_HEADER_RANDWRITER_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS6_HEADER_RANDWRITER_HPP_

#include <iostream>
#include <string>
using std::string;
#include <map>
#include <utility>
using std::pair;
#include <chrono>
#include <random>
#include <algorithm>

class RandWriter {
 public:
    // Note: all of the below constructors/methods should be public.
    // -------------------------------------------------------------------
    // create a Markov model of order k from given text
    RandWriter(string text, int k);  // Assume that text has length at least k.
    //  -------------------------------------------------------------------
    int orderK() const;  // order k of Markov model
    // -------------------------------------------------------------------
    // number of occurrences of kgram in text
    int freq(string kgram) const;  // throw an exception if kgram is not of
    // length k
    // ------------------------------------------------------------------
    // number of times that character c follows kgram
    // if order=0, return num of times char c appears
    // (throw an exception if kgram is not of length k)
    int freq(string kgram, char c) const;
    // ------------------------------------------------------------------
    // random character following given kgram
    // (Throw an exception if kgram is not of length k.
    // Throw an exception if no such kgram.)
    char kRand(string kgram);
    // ------------------------------------------------------------------
    // generate a string of length L characters
    // by simulating a trajectory through the corresponding
    // Markov chain. The first k characters of the newly
    // generated string should be the argument kgram.
    // Throw an exception if kgram is not of length k.
    // Assume that L is at least k.
    string generate(string kgram, int L);
    // ------------------------------------------------------------------
    // overload the stream insertion operator and display
    std::ostream& operator<<(std::ostream& out);
    // the internal state of the Markov Model. Print out
    // the order, the alphabet, and the frequencies of
    // the k-grams and k+1-grams.4

 private:
    int order;
    string input;
    string alphabet;
    string currState;
    // map to hold k grams and the maps it to K+1 grams and there occurence
    std::map<string, int> kfreq;
    std::map<string, std::map<string, int>> symbolTable;
};
#endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS6_HEADER_RANDWRITER_HPP_


```

RandWriter.cpp

```
// Copyright [2023] <Daniel Bergeron>
#include "Header/RandWriter.hpp"

template<class Function>
void loadHeader(Function fn);

// Note: all of the below constructors/methods should be public.
    // ------------------------------------------------------------------
    // create a Markov model of order k from given text
    // Assume that text has length at least k.
```

```
RandWriter::RandWriter(string text, int k) {
    input = text;
    currState = text;
    int size = static_cast<int>(text.size());
    order = k;
    //  make alpha bet
    char c;
    alphabet += text[0];
    for (int i = 1; i < size; i++) {
        c = text[i];
        // find returns npos -1 if nothing found
        if (static_cast<int>(alphabet.find(c, 0)) < 0) {
            alphabet += c;
        }
    }

    std::string kgram, temp;
    std::map<string, int> kGramPlus1;
    // loop to fill symbol table
    // i increments the entire sting
    // j increments the kgram on the string
    int j = 0;
    int index = 0;

    for (int i = 0; i < size; i++) {
        kgram.clear();
        index = i;
        for (j = 0; j < order; j++) {
            if (index > size - 1) {
                index = 0;
                kgram += text[index];
            } else {
                kgram += text[index];
            }
            index++;
        }
        // insert kgram into its text freq map
        kfreq[kgram]++;
        // we have a kgram now we must break it into
        // its k + 1
        // builds new map with value and ties it to
        // old map
        symbolTable[kgram];
        if (index > size - 1) {
            index = 0;
        }
        temp = kgram;
        temp += text[index];

        kGramPlus1[temp] = 0;
        (symbolTable[kgram])[temp]++;
    }
}

int RandWriter::orderK() const {  // order k of Markov model
    return order;
}

// -------------------------------------------------------------------
    // number of occurrences of kgram in text
    // throw an exception if kgram is not of length k
int RandWriter::freq(string kgram) const {
    int size = 0;
    size = static_cast<int>(kgram.length());
    int freq = 0;
    if (order == 0 && kgram == " ") {
        return static_cast<int>(input.size());
    }
    if (size != order) {
        std::runtime_error error(string("Size Mismatch"));
        throw error;
    }
    if (kfreq.find(kgram) == kfreq.end()) {
        return 0;
    }
    freq = kfreq.find(kgram)->second;
    return freq;
}

// number of times that character c follows kgram
// if order=0, return num of times char c appears
// (throw an exception if kgram is not of length k)
int RandWriter::freq(string kgram, char c) const {
    int count = 0;
    int size = 0;
    size = static_cast<int>(kgram.length());
    if (size != order) {
        std::runtime_error error(string("Size Mismatch"));
        throw error;
    }
    string p1kgram;  // k + 1
    // without lamda
    if (order == 0) {
        for (int i = 0; i < static_cast<int>(input.size()); i++) {
            if (input[i] == c) {
                count++;
            }
        }
        return count;
    }
```

```
        // with lambda
        // we need to kgram plus one
        if (kfreq.find(kgram) == kfreq.end()) {
            return 0;
        }
        auto n = symbolTable.find(kgram)->second.begin();
        for (; n != symbolTable.find(kgram)->second.end(); n++) {
            p1kgram = n->first;
            // if first char == c increase the count
            if (p1kgram[static_cast<int>((p1kgram.size()-1))] == c) {
                count += n->second;
            }
        }
        return count;
    }

    // ------------------------------------------------------------------
    // random character following given kgram
    // (Throw an exception if kgram is not of length k.
    // Throw an exception if no such kgram.)
    char RandWriter::kRand(string kgram) {
        // get all k plus one grams there occurences
        // add occurences up
        // generate random number of occurences and from 0 to total occurences
        // put number on range if
        // construct a trivial random generator engine from a time-based seed:
        int size = static_cast<int>(kgram.length());
        if (size != order) {
            std::runtime_error error(string("Size Mismatch"));
            throw error;
        }
        if (kfreq.find(kgram) == kfreq.end()) {
            std::runtime_error error(string("Invalid Kgram"));
            throw error;
        }
        // create generator
        unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
        std::default_random_engine generator(seed);
        auto p = symbolTable[kgram].begin();
        int upperBounds = 0, randNum;

        // gets total amount for upperbounds
        for (unsigned i = 0; i < alphabet.size(); i++) {
            upperBounds += freq(kgram, alphabet[i]);
        }

        // generate number
        randNum = (generator() % upperBounds) + 1;

        int first = 0, second = 0;
        char c = kgram[0];
        // use randnum to get next letter out of range of possible next char
        for (p = symbolTable[kgram].begin(); p != symbolTable[kgram].end(); p++) {
            second += p->second;
            // std::cout << "First: " << first << "Second: " << second << std::endl;
            if (randNum <= second) {
                c = p->first[size];
                break;
            } else {
                first += p->second;
            }
        }
        return c;
    }

    // -------------------------------------------------------------------
    // generate a string of length L characters
    // by simulating a trajectory through the corresponding
    // Markov chain. The first k characters of the newly
    // generated string should be the argument kgram.
    // Throw an exception if kgram is not of length k.
    // Assume that L is at least k.

    string RandWriter::generate(string kgram, int L) {
        // first get kgram size
        string chain;
        int size = static_cast<int>(kgram.length());
        if (size != order) {
            std::runtime_error error(string("Size Mismatch"));
            throw error;
        }
        if (kfreq.find(kgram) == kfreq.end()) {
            std::runtime_error error(string("Invalid Kgram"));
            throw error;
        }
        if (size > L) {
            for (int i = 0; i < L; i++) {
                chain += kgram[i];
            }
            currState = chain;
            return chain;
        }
        // else we begin the generation

        chain = kgram;
        char c;
        int i = size;
```



```cpp
        // std::cout << size-1 << std::endl;
        // gets random character from the kgrams
        while (i < L) {
            c = kRand(kgram);
            chain += c;
            kgram.replace(0, size, chain, i - (size - 1), size);
            i++;
        }
        currState = chain;
        return chain;
    }

    // -------------------------------------------------------------------
    // << // overload the stream insertion operator and display
    // the internal state of the Markov Model. Print out
    // the order, the alphabet, and the frequencies of
    // the k-grams and k+1-grams.
    std::ostream& RandWriter::operator<<(std::ostream& out) {
        auto print = [&]() {
            out << "\nOrder: " << order << std::endl;
            out << "Alphabet:\n" << alphabet << std::endl;
            out << "CurrState:\n" << currState << std::endl;
            out << "Frequcies k-grams:\n" << std::endl;
        };

        loadHeader(print);

        for (auto i : symbolTable) {
            out << i.first << " " << freq(i.first) << std::endl;
        }

        out << "Frequcies k+1-grams:\n" << std::endl;
        string str;
        for (auto i : symbolTable) {
            for (auto j : i.second) {
                str = j.first;
                out << str << " " << j.second << std::endl;
            }
        }
        return out;
    }

    // lambda practice
    template<class Function>
    void loadHeader(Function fn) {
        fn();
    }

```

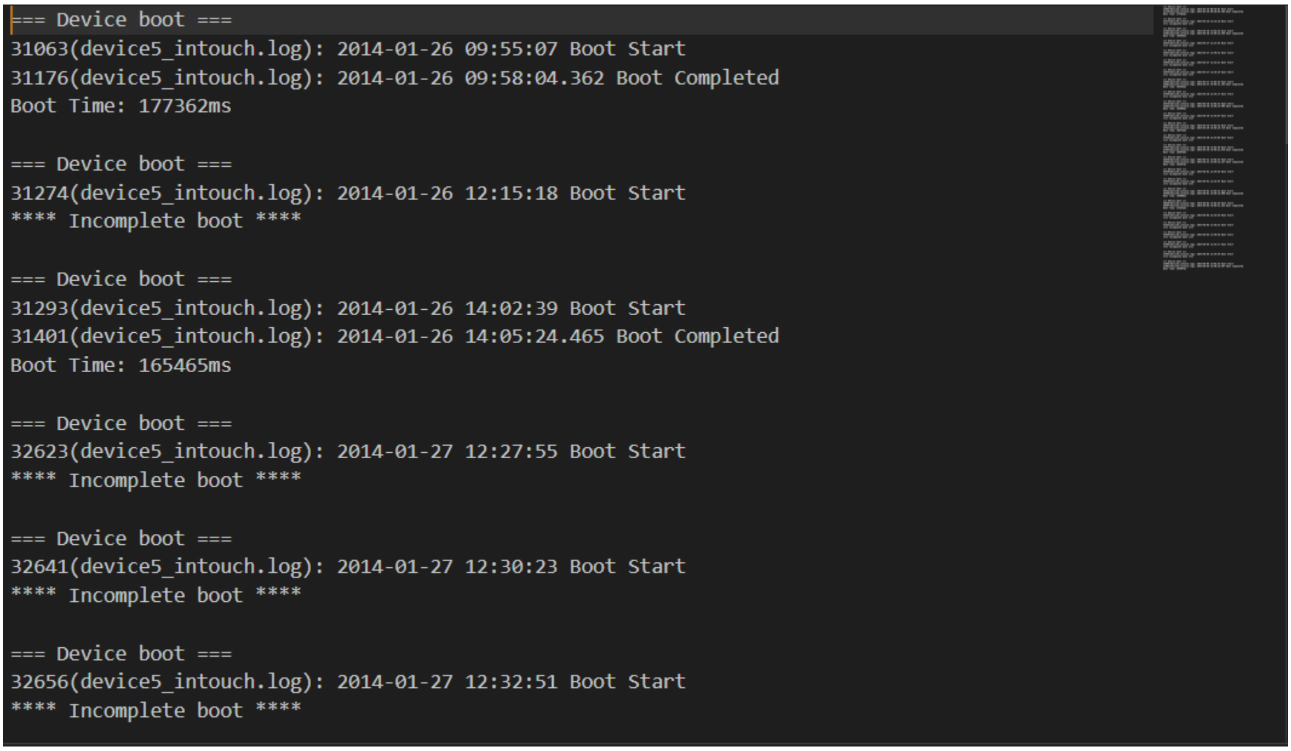
Test.cpp

```cpp
// Copyright [2023] <Daniel Bergeron>
#include "Header/RandWriter.hpp"

#define BOOST_TEST_DYN_LINK
#define BOOST_TEST_MODULE TEST
#include <boost/test/unit_test.hpp>

// tests exception for size mismatchs and if it is the wrong kgram
BOOST_AUTO_TEST_CASE(exceptionThrows) {
    RandWriter test("gagggagaggcgagaaa", 2);
    BOOST_REQUIRE_THROW(test.freq("gdg"), std::runtime_error);
    BOOST_REQUIRE_THROW(test.freq("gdg", 'c'), std::runtime_error);
    BOOST_REQUIRE_THROW(test.kRand("gdg"), std::runtime_error);
    BOOST_REQUIRE_THROW(test.kRand("fr"), std::runtime_error);
    BOOST_REQUIRE_THROW(test.kRand("gdgcd"), std::runtime_error);
    BOOST_REQUIRE_THROW(test.generate("gdg", 1), std::runtime_error);
    BOOST_REQUIRE_THROW(test.generate("eg", 1), std::runtime_error);
}

BOOST_AUTO_TEST_CASE(exceptionNoThrow) {
    RandWriter test("gagggagaggcgagaaa", 2);
    BOOST_REQUIRE_NO_THROW(test.freq("gg"));
    BOOST_REQUIRE_NO_THROW(test.freq("gg", 'c'));
    BOOST_REQUIRE_NO_THROW(test.kRand("gg"));
    BOOST_REQUIRE_NO_THROW(test.kRand("gc"));
    BOOST_REQUIRE_NO_THROW(test.generate("ga", 100));
}

BOOST_AUTO_TEST_CASE(methodAccuracyTests) {
    RandWriter test("gagggagaggcgagaaa", 2);
    BOOST_REQUIRE(test.orderK() == 2);
    BOOST_REQUIRE(test.freq("gg") == 3);
    BOOST_REQUIRE(test.freq("ff") == 0);
    BOOST_REQUIRE(test.freq("gg", 'c') == 1);
    BOOST_REQUIRE(test.kRand("cg") == 'a');
    string str;
    str = test.generate("gg", 5);
    BOOST_REQUIRE(str.length() == 5);
}

BOOST_AUTO_TEST_CASE(orderZeroTest) {
    string str = "gagggagaggcgagaaa";
    RandWriter test(str, 0);
    BOOST_REQUIRE(test.orderK() == 0);
    BOOST_REQUIRE(test.freq("", 'c') == 1);
    BOOST_REQUIRE(test.kRand(""));
    str = test.generate("", 5);
    BOOST_REQUIRE(str.length() == 5);
}
```

PS7 – Kronos Time Clock

Program Discussion

This program was my formal introduction to working with regular expression. Regular expression is a way a programmer can parse strings and build substrings with the data they desire. This was one of the more trivial assignments of this semester and did not require much. This program took input from a Kronos Device log file and used regular expression to pull out information about successful device boots. You are then required to put that information into a .rpt file which only contains successful starts and line number accompanied by if it failed or succeeded. After that the .rpt file contains a time stamp for when each of these events succeeded or failed. The main difficulty with this program for me didn't come from the regular expression but the managing the flags that tell if the program has already run into a start or not. This was difficult logic to construct, but surprisingly using a more descriptive name for the flag variable helped me think it out. Other than that, integrating the boost time library was a little difficult but by reading the documentation I was able to figure it out.



```
=== Device boot ===
31063(device5_intouch.log): 2014-01-26 09:55:07 Boot Start
31176(device5_intouch.log): 2014-01-26 09:58:04.362 Boot Completed
Boot Time: 177362ms

=== Device boot ===
31274(device5_intouch.log): 2014-01-26 12:15:18 Boot Start
**** Incomplete boot ****

=== Device boot ===
31293(device5_intouch.log): 2014-01-26 14:02:39 Boot Start
31401(device5_intouch.log): 2014-01-26 14:05:24.465 Boot Completed
Boot Time: 165465ms

=== Device boot ===
32623(device5_intouch.log): 2014-01-27 12:27:55 Boot Start
**** Incomplete boot ****

=== Device boot ===
32641(device5_intouch.log): 2014-01-27 12:30:23 Boot Start
**** Incomplete boot ****

=== Device boot ===
32656(device5_intouch.log): 2014-01-27 12:32:51 Boot Start
**** Incomplete boot ****
```

File Output 1

Key Elements Discussion

The heart of this program was regular expression, and the boost time library. These Two components are what built this program. After these two features were implemented, it just became a trivial I'll be it slightly frustrating programming problem.

What I Learned

Despite doing very little in the way of work for this problem I learned two very important things. First how to use regular expression, which is a very powerful tool I don't plan on putting down. In handling strings, it is unmatched, many possibilities have opened to me because I learned this skill. Going forward Ill be able to write programs that can parse huge text files and find information I need with little effort. Before this I had to go character by character or do string comparisons. This skill is going to save me a great deal of time in future endeavors. The second major thing I learned from this program is the importance of reading documentation. This program would have been very difficult to implement on my own if I had not implemented libraries. These libraries, while convenient, need to be understood to be utilized to their fullest potential. The only what to do that is to read the documentation.

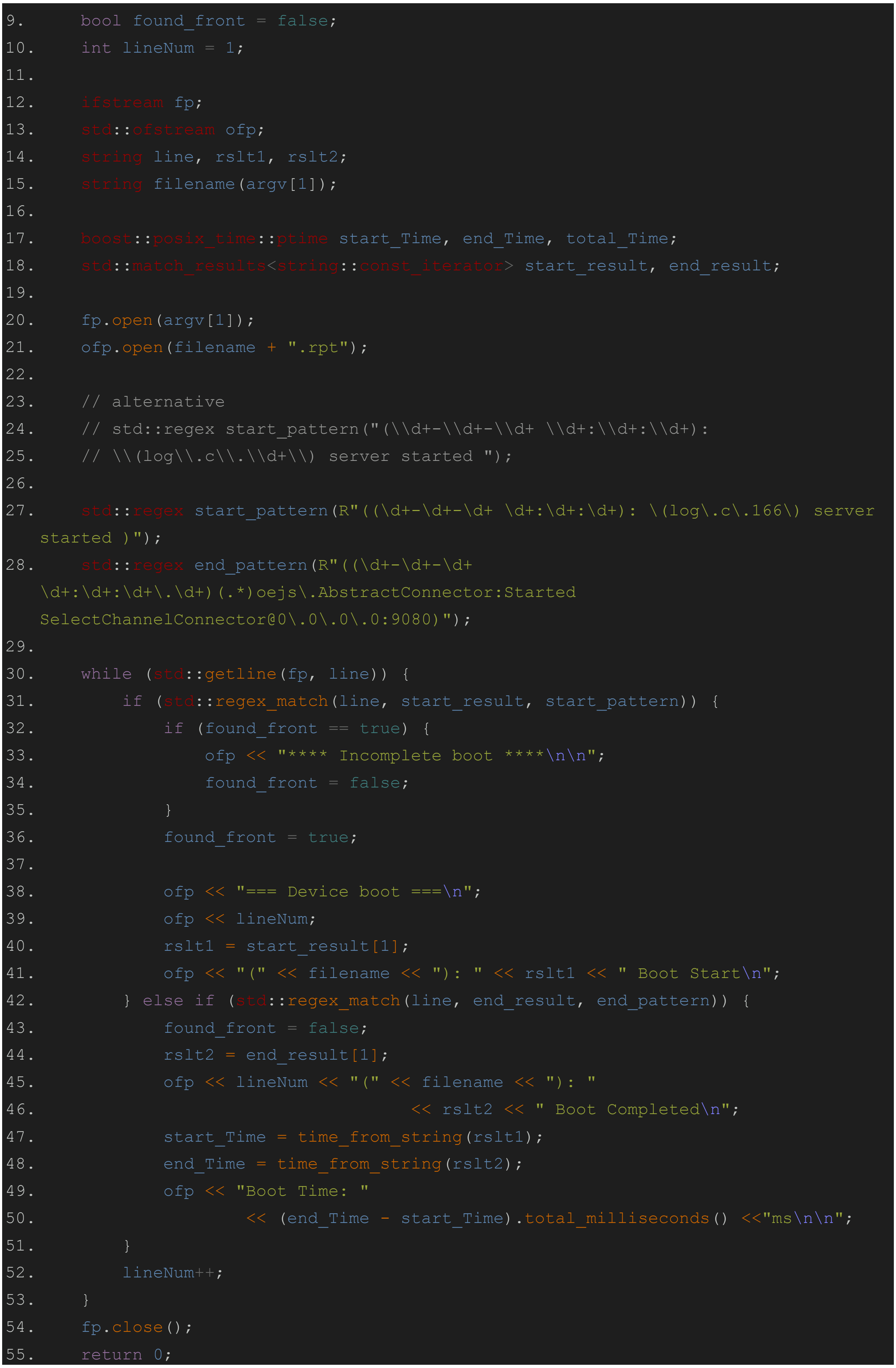
Code

Makefile

```makefile
CC = g++
CFLAGS = -std=c++11 -g -Og -Wall -Werror -pedantic
OBJ = main.o
LIBS = -lboost_regex -lboost_date_time
DEPS = Header/KronosLog.hpp

all: ps7 lint

ps7: $(OBJ)
	$(CC) $(CFLAGS) -o ps7 $(OBJ) $(LIBS)
lint:
	cpplint *.cpp $(DEPS)
%.o: %.cpp $(DEPS)
	$(CC) $(CFLAGS) -c $<
clean:
	rm -f $(OBJ) lint ps7

```

main.cpp

```cpp
// Copyright [2023] <Daniel Bergeron>
#include "Header/KronosLog.hpp"

int main(int argc, char* argv[]) {
    if (argc != 2) {
        cout << "Wrong number of arguments\n";
        return -1;
    }
```



```cpp
    bool found_front = false;
    int lineNum = 1;

    ifstream fp;
    std::ofstream ofp;
    string line, rslt1, rslt2;
    string filename(argv[1]);

    boost::posix_time::ptime start_Time, end_Time, total_Time;
    std::match_results<string::const_iterator> start_result, end_result;

    fp.open(argv[1]);
    ofp.open(filename + ".rpt");

    // alternative
    // std::regex start_pattern("(\\d+-\\d+-\\d+ \\d+:\\d+:\\d+):
    // \\(log\\.c\\.\\d+\\) server started ");

    std::regex start_pattern(R"((\d+-\d+-\d+ \d+:\d+:\d+): \(log\.c\.166\) server started )");
    std::regex end_pattern(R"((\d+-\d+-\d+ \d+:\d+:\d+\.\d+)(.*)oejs\.AbstractConnector:Started SelectChannelConnector@0\.0\.0\.0:9080)");

    while (std::getline(fp, line)) {
        if (std::regex_match(line, start_result, start_pattern)) {
            if (found_front == true) {
                ofp << "**** Incomplete boot ****\n\n";
                found_front = false;
            }
            found_front = true;

            ofp << "=== Device boot ===\n";
            ofp << lineNum;
            rslt1 = start_result[1];
            ofp << "(" << filename << "): " << rslt1 << " Boot Start\n";
        } else if (std::regex_match(line, end_result, end_pattern)) {
            found_front = false;
            rslt2 = end_result[1];
            ofp << lineNum << "(" << filename << "): "
                                    << rslt2 << " Boot Completed\n";
            start_Time = time_from_string(rslt1);
            end_Time = time_from_string(rslt2);
            ofp << "Boot Time: "
                    << (end_Time - start_Time).total_milliseconds() <<"ms\n\n";
        }
        lineNum++;
    }
    fp.close();
    return 0;
```

```
}


```

KronosLog.hpp

```
// Copyright [2023] <Daniel Bergeron>
#ifndef _HOME_DBERGERO3_PROGRAMS_COMP4_PS7_HEADER_KRONOSLOG_HPP_
#define _HOME_DBERGERO3_PROGRAMS_COMP4_PS7_HEADER_KRONOSLOG_HPP_

#include <iostream>
using std::cin;
using std::cout;
using std::endl;
#include <string>
using std::string;
#include <fstream>
using std::ifstream;
#include <regex>
#include <boost/regex.hpp>
#include "boost/date_time/posix_time/posix_time.hpp"
using boost::posix_time::time_from_string;

#endif  // _HOME_DBERGERO3_PROGRAMS_COMP4_PS7_HEADER_KRONOSLOG_HPP_


```